

Bachelorarbeit

Regret- k -Insertion für das statische linienbasierte Dial-a-Ride-Problem

Albert Hörner

Abgabedatum: 28. Juli 2026
Betreuer: Prof. Dr. Marie Schmidt
Kendra Reiter



Julius-Maximilians-Universität Würzburg
Lehrstuhl für Informatik I
Algorithmen und Komplexität

Zusammenfassung

Das linienbasierte Dial-a-Ride-Problem (liDARP) ist eine Variante des Dial-a-Ride-Problems, bei der Fahrzeuge nachfragebasiert auf einer vordefinierten Linie operieren und dabei zusätzliche Flexibilität gegenüber einem klassischen Linienbus haben. Für die exakte Lösung des statischen liDARP existiert eine ereignisbasierte MILP-Formulierung, deren Rechenzeit auf größeren Instanzen jedoch stark ansteigt.

In dieser Arbeit wird eine Regret- k -Insertion-Heuristik für das statische liDARP entwickelt, implementiert und evaluiert. Statt in jedem Schritt die Anfrage mit der günstigsten Einfügeoption einzuplanen, priorisiert das Verfahren jene Anfrage, deren Aufschub den größten späteren Verlust bedeuten würde; dieser Regret-Wert wird über die k besten Einfügeoptionen je Fahrzeug berechnet. Verglichen wird gegen eine Greedy-Baseline und gegen die exakte Referenz (Gurobi, Zeitlimit 1800 Sekunden) auf den 14 etablierten Benchmark-Instanzen (16 bis 58 Anfragen) sowie auf 19 erweiterten Instanzen mit bis zu 132 Anfragen.

Die Heuristik bedient auf 32 der 33 untersuchten Instanzen sämtliche Anfragen, während die Greedy-Baseline auf mehreren Instanzen Anfragen ablehnt. Ihr Abstand zum Zielfunktionswert beträgt auf den exakt gelösten Instanzen im Mittel 17,2 % und bleibt, konservativ gegen bewiesene Schranken gemessen, auch auf den größeren Instanzen stabil (im Mittel 14,4 bzw. 15,2 %); er entsteht fast ausschließlich aus der Routeneffizienz, nicht aus Ablehnungen. Dafür rechnet die Heuristik in Sekundenbruchteilen bis wenigen Sekunden, während das MILP auf gängiger Konsumentenhardware (Intel Core i7 der zwölften Generation) ab etwa 36 Anfragen das Zeitlimit erreicht, ab etwa 84 Anfragen zunehmend Anfragen unbedient lässt und auf der größten Instanz keine Lösung mehr liefert. Der Einfluss des Parameters k auf die Lösungsqualität bleibt dabei gering und uneinheitlich. Regret- k eignet sich damit als schnelles Verfahren, das das Erschließungsziel des liDARP praktisch vollständig erfüllt; die Kilometerersparnis des Ridepoolings erreicht es nur eingeschränkt.

Inhaltsverzeichnis

1	Einleitung	5
1.1	Motivation	5
1.2	Zielsetzung der Arbeit	6
2	Problembeschreibung	7
2.1	Einordnung in die Familie der Routenplanungsprobleme	7
2.2	Das Dial-a-Ride-Problem (DARP)	8
2.3	Das linienbasierte DARP (liDARP)	9
2.4	Mathematische Definition des statischen liDARP	10
3	Verwandte Arbeiten	13
3.1	Exakte Lösungsverfahren und Komplexität des liDARP	13
3.2	Tabu-Suche für das dynamische linienbasierte DARP	14
3.3	Regret-Heuristiken in der Vehicle-Routing-Literatur	17
4	Methodik	20
4.1	Gemeinsames Grundgerüst der Insertion-Heuristiken	20
4.1.1	Lösungsrepräsentation	20
4.1.2	Zulässigkeitsbedingungen	21
4.1.3	Einfügekosten	22
4.2	Greedy-Insertion als Baseline	23
4.3	Regret- k -Insertion	24
4.3.1	Grundprinzip und Regret-Wert	24
4.3.2	Zulässige Einfügepositionen im liDARP	25
4.3.3	Einfügekosten und Zielfunktionsbeitrag	25
4.4	Algorithmische Gesamtübersicht	26
5	Implementierung	29
5.1	Kernfragen und Lösungen	30
5.1.1	Struktur	30
5.1.2	Leerfahrten	31
5.1.3	Depot	32
5.1.4	Initialisierung	33
5.1.5	Einfügepositionen und beste Einfügeposition	35
5.1.6	Verlängern einer Sublinie und die Wirkung auf die Nachbarn	36
5.1.7	Annahme der Anfrage	38
5.1.8	Regret- k -Kosten	39

5.2	Datenstrukturen	41
5.2.1	Request	42
5.2.2	Instance	43
5.2.3	Stop	44
5.2.4	Subline	45
5.2.5	VehicleRoute	46
5.2.6	Solution	47
5.2.7	InsertionCandidate	48
5.2.8	Zusammenfassung der Hierarchie	48
5.3	Methoden	50
5.3.1	Gültige Positionen finden	51
5.3.2	Einfügekosten über die Stop-Folge	51
6	Evaluation	54
6.1	Versuchsaufbau	54
6.2	Ergebnisse und Wahl der Seeding-Strategie	55
6.3	Exakte Referenzlösungen (Event-Based-MILP)	61
6.4	Vergleich	63
6.5	Erweiterte Instanzen	65
6.6	Einfluss der Gewichtsparameter	66
7	Diskussion	69
7.1	Myopie: gelöst bei der Auswahl, offen bei der Platzierung	69
7.2	Einordnung der Ergebnisse	70
7.3	Limitationen	71
8	Fazit und Ausblick	72
8.1	Fazit	72
8.2	Ausblick	73
	Literaturverzeichnis	74

1 Einleitung

1.1 Motivation

Klassische Linienbusse fahren feste Routen nach festen Fahrplänen. Das ist für hohe Nachfragedichten effizient, aber in dünn besiedelten Gebieten oder zu Randzeiten oft unwirtschaftlich: Busse fahren leer, Fahrgäste warten lang, oder bestimmte Strecken werden gar nicht bedient [RSS24]. On-Demand-Mobilitätskonzepte versuchen, diese Lücke zu schließen, indem sie nachfragebasiert routen. Der Preis dafür ist eine deutlich höhere Komplexität in der Fahrzeugdisposition. Das linienbasierte Dial-a-Ride-Problem (liDARP) bildet einen Mittelweg ab [RSS24]: Fahrzeuge bedienen Anfragen flexibel, sind dabei aber an eine vordefinierte Linie gebunden. Sie können entlang der Linie warten, abkürzen oder die Richtung wechseln, dürfen jedoch nur leer wenden [RSS24, LRS25]. Diese Struktur reduziert die Komplexität gegenüber einem unbeschränkten DARP erheblich, ohne die Vorteile flexibler Bedienung aufzugeben. Für die optimale Lösung des statischen liDARP existieren MILP-Formulierungen [RSS24], für die Reiter et al. Laufzeiten von unter zehn Sekunden für Instanzen mit bis zu 132 Anfragen berichten. Bei realistischen Instanzgrößen (mehreren Dutzend bis Hunderten Anfragen) steigen die Rechenzeiten exakter Verfahren jedoch schnell stark an. Heuristiken bieten hier einen Ausweg: Sie liefern in kurzer Zeit zulässige Lösungen, deren Qualität allerdings je nach Verfahren stark schwanken kann [DD04]. Eine spezielle Klasse von Heuristiken sind Insertion-Verfahren, bei denen Anfragen sequenziell in die bestehenden Routen eingefügt werden. Greedy-Insertion entscheidet dabei in jedem Schritt lokal optimal, ohne die Konsequenzen für später noch nicht eingeplante Anfragen zu berücksichtigen, ein als *myopisch* bekanntes Problem. Regret- k -Insertion versucht, diesen Nachteil zu kompensieren, indem sie nicht die Anfrage mit der besten Einfügeoption priorisiert, sondern jene, für die ein Aufschub am teuersten würde [RP06, PR07]. Während Regret-Insertion für klassische Vehicle-Routing-Probleme [PR07, RP06] und das ungebundene DARP [DD04] gut untersucht ist, existieren bislang keine Studien zur Anwendung auf das linienbasierte DARP. Diese Arbeit schließt diese Lücke: Sie entwickelt eine Regret- k -Insertion-Heuristik für das statische liDARP und vergleicht sie auf etablierten Benchmark-Instanzen sowohl gegen einen Greedy-Baseline-Ansatz als auch gegen die exakte MILP-Lösung [RSS24]. Statisch bedeutet dabei, dass alle Anfragen bereits vor Beginn der Planung bekannt sind. Im Fokus steht dabei die Frage, wie nahe eine konstruktive Heuristik an das Optimum herankommt und ab welcher Instanzgröße ihre Geschwindigkeitsvorteile ins Gewicht fallen.

1.2 Zielsetzung der Arbeit

Ziel dieser Arbeit ist die Entwicklung und Evaluation einer Regret- k -Insertion-Heuristik für das statische linienbasierte Dial-a-Ride-Problem (liDARP). Die Heuristik wird auf etablierten Benchmark-Instanzen [RSS24] unterschiedlicher Größe ausgeführt und systematisch mit zwei Referenzverfahren verglichen: einer Greedy-Insertion als heuristischer Baseline und der exakten MILP-Lösung [RSS24] als optimalem Vergleichswert auf Instanzen, die noch in akzeptabler Zeit lösbar sind.

Der Vergleich erfolgt anhand mehrerer Qualitätsparameter, die die Lösungsgüte sowohl aus Sicht der Fahrgäste als auch aus Sicht des Betreibers abbilden. Bewertet werden insbesondere:

- der *Zielfunktionswert*: die gewichtete Kombination aus beförderten Fahrgästen und eingesparter Distanz,
- die *Akzeptanzrate*, also der Anteil der bedienten Anfragen,
- die *Optimalitätslücke* gegenüber der MILP-Lösung,
- die *Berechnungszeit* der Heuristik.

Diese Metriken erlauben eine differenzierte Aussage darüber, wie nah Regret- k an das Optimum herankommt und ab welcher Instanzgröße sich die Geschwindigkeitsvorteile der Heuristik gegenüber dem MILP-Solver auszahlen. Zudem wird der Einfluss des Parameters k auf das Verhältnis von Lösungsqualität und Laufzeit untersucht.

Eine Beobachtung aus verwandten Arbeiten am Lehrstuhl ist, dass sich Fahrzeuge bei rein lokal optimierenden Strategien über die Zeit räumlich auf einer Seite der Linie sammeln können, was die Bedienung späterer Anfragen aus dem entgegengesetzten Bereich erschwert. Da Regret- k durch die Berücksichtigung von Opportunitätskosten weniger *myopisch* arbeitet als ein klassischer Greedy-Ansatz [RP06, DD04], liegt die Vermutung nahe, dass diese Eigenschaft auch im liDARP zu einer ausgewogeneren Verteilung der Fahrzeuge entlang der Linie beiträgt; eine eigene Untersuchung dieser Verteilungsfrage liegt außerhalb des Rahmens dieser Arbeit.

2 Problembeschreibung

2.1 Einordnung in die Familie der Routenplanungsprobleme

Das liDARP ist eine Variante innerhalb der breiten Klasse der *Vehicle-Routing-Probleme* (VRP), die allgemein die kostenminimale Routung einer Fahrzeugflotte zur Bedienung räumlich verteilter Anfragen zum Gegenstand haben [LMR⁺26]. Innerhalb des VRP bildet das *Pickup-and-Delivery-Problem* (PDP) eine Spezialisierung durch zwei strukturelle Zusatzbedingungen [LMR⁺26, PDH08]:

- **Vorrang** (*precedence*): jede Anfrage besitzt einen Abhol- und einen Abgabepunkt; der Abholpunkt muss vor dem Abgabepunkt angefahren werden.
- **Paarung** (*pairing*): beide Punkte einer Anfrage müssen vom selben Fahrzeug bedient werden.

Innerhalb des PDP wird zwischen Anwendungen für Güter und für Personen unterschieden [LMR⁺26, PDH08, CL07]: Werden Güter befördert, behält das Problem die Bezeichnung PDP; werden Personen befördert und treten nutzerorientierte Servicebedingungen (Fahrzeit-, Wartezeitobergrenzen) hinzu, spricht man vom *Dial-a-Ride-Problem* (DARP).

Das *linienbasierte Dial-a-Ride-Problem* (liDARP) ist schließlich eine Variante des DARP, in der die Fahrzeugbewegungen räumlich an eine vordefinierte *Linie* gebunden sind: eine feste, geordnete Folge von Haltestellen, die von allen Fahrzeugen gemeinsam bedient und entlang dieser Reihenfolge in einer der beiden Richtungen (aufsteigend oder absteigend) befahren wird, wobei Richtungswechsel nur in leerem Zustand erlaubt sind [RSS24, LMR⁺26]. Insgesamt ergibt sich damit die Hierarchie

$$\text{VRP} \supset \text{PDP} \supset \text{DARP} \supset \text{liDARP},$$

in der jede Stufe die vorherige um zusätzliche strukturelle oder serviceorientierte Bedingungen verschärft. Eine ausführliche Klassifikation der Pickup-and-Delivery-Probleme inklusive ihrer Untergliederung in gepaarte und ungepaarte Varianten findet sich bei Parragh et al. [PDH08]. Die Einführung des liDARP als linienbasierte Variante geht auf Reiter, Schmidt und Stiglmayr [RSS24] zurück.

Aus Sicht der Verkehrsplanung lässt sich diese Hierarchie um eine zweite, betriebswirtschaftliche Achse ergänzen [LMR⁺26]: Klassische Linienbusse sind räumlich und zeitlich *starr*, vollständige On-Demand-Systeme (DARP) sind in beiden Dimensionen *flexibel*. Das liDARP nimmt eine Zwischenstellung als *semi-flexibles* System ein, das räumlich an die Linie gebunden, zeitlich aber flexibel ist und damit zwischen den beiden Extremen vermittelt.

Die folgenden Abschnitte vertiefen diese Hierarchie schrittweise: Abschnitt 2.2 beschreibt das DARP als nächstliegende Oberklasse, Abschnitt 2.3 führt das liDARP als linienbasierte Spezialisierung ein, und Abschnitt 2.4 gibt die mathematische Definition des in dieser Arbeit betrachteten statischen liDARP.

2.2 Das Dial-a-Ride-Problem (DARP)

Das Dial-a-Ride-Problem (DARP) befasst sich mit der Gestaltung von Fahrzeugrouten und Abfahrtszeiten für Nutzer, die individuelle Transportanfragen zwischen einem Abholpunkt und einem Ablieferungspunkt stellen [CL07]. Ziel ist es, mit einer Flotte von m Fahrzeugen möglichst viele Anfragen unter Einhaltung betrieblicher und servicebezogener Nebenbedingungen zu bedienen. Das klassische Anwendungsszenario ist die Tür-zu-Tür-Beförderung älterer oder mobilitätseingeschränkter Personen, die reguläre Linienbusse nicht oder nur eingeschränkt nutzen können [CL07, HSK⁺18]. Angesichts einer alternden Bevölkerung gewinnen Dial-a-Ride-Systeme dabei nicht nur verkehrlich, sondern auch gesellschaftlich an Bedeutung, da sie soziale Isolation gefährdeter Bevölkerungsgruppen verringern können [MBC17].

Was das DARP von klassischen Fahrzeugroutierungsproblemen wie dem Vehicle Routing Problem (VRP) unterscheidet, ist die zentrale Bedeutung der Fahrgastperspektive [CL07]: Die Minimierung von Nutzerunbequemlichkeit (insbesondere Wartezeit am Abholpunkt und Fahrtzeit im Fahrzeug) muss gegen die Minimierung der Betriebskosten abgewogen werden. Beide Ziele können in Konflikt stehen. Die häufigsten Optimierungsziele umfassen Betriebskosten (Gesamtfahrstrecke, Fahrzeuganzahl, Routendauer) sowie nutzerorientierte Kenngrößen (Wartezeit, Fahrtzeit, Abweichung vom Wunschzeitpunkt) [HSK⁺18, MBC17].

Zu den typischen Strukturmerkmalen des DARP zählen [HSK⁺18]: Zeitfenster für Abholung und Ablieferung, eine maximale Fahrtzeit pro Anfrage (häufig in Relation zur Direktfahrtzeit), eine Fahrzeugkapazität sowie eine maximale Routendauer pro Fahrzeug. Ho et al. [HSK⁺18] klassifizieren DARP-Varianten entlang zweier Dimensionen: ob während der Ausführung neue Information auftaucht (*statisch* vs. *dynamisch*) und ob die vorhandene Information sicher oder mit Unsicherheit behaftet ist (*deterministisch* vs. *stochastisch*). In dieser Arbeit wird ausschließlich die statisch-deterministische Variante betrachtet, bei der alle Anfragen vor Beginn der Planung vollständig und sicher bekannt sind.

Das DARP verallgemeinert das Vehicle Routing Problem with Time Windows (VRPTW) und das Pickup-and-Delivery-Problem [CL07]; es ist NP-schwer [MBC17]. Exakte Lösungsverfahren stoßen daher bei realistischen Instanzgrößen schnell an ihre Grenzen, weshalb heuristische und metaheuristische Ansätze in der Literatur breiten Raum einnehmen: von einfachen Insertion-Heuristiken über Tabu Search und Large Neighborhood Search bis hin zu Branch-and-Cut-Verfahren auf Basis spezieller MILP-Formulierungen [CL07].

Seit etwa 2007 ist ein deutlich gestiegenes Forschungsinteresse am DARP zu beobachten [HSK⁺18]. Als treibende Kräfte gelten einerseits gesellschaftliche Entwicklungen wie die Sharing Economy und das Konzept *Mobility as a Service*, das eine Abkehr vom

Fahrzeugbesitz hin zu bedarfsgerechter Mobilität propagiert, andererseits technologische Fortschritte in den Bereichen mobile Kommunikation, Cloud Computing und Datenanalytik, die neue Betriebsformen erst ermöglichen [HSK⁺18]. Hinzu kommen Umweltmotive: Ridepooling-Systeme können den CO₂-Fußabdruck des Individualverkehrs senken, indem sie Fahrgäste mit ähnlichen Routen in einem Fahrzeug bündeln [HSK⁺18]. Diese Entwicklungen machen das DARP und seine Varianten (darunter das linienbasierte DARP) zu einem aktiven Forschungsgebiet mit wachsender praktischer Relevanz.

Das DARP wird in der Literatur bereits seit den 1970er-Jahren behandelt; eine frühe Arbeit ist etwa Stein [Ste77], der eine asymptotische Analyse von Scheduling-Verfahren für Dial-a-Ride-Systeme vorlegte. Eine systematische Zusammenstellung formaler Modelle und Algorithmen für das DARP findet sich bei Cordeau und Laporte [CL07]. Umfassende Überblicke über Problemvarianten und Lösungsansätze finden sich zudem in den Surveys von Ho et al. [HSK⁺18] und Molenbruch et al. [MBC17].

2.3 Das linienbasierte DARP (liDARP)

Das linienbasierte Dial-a-Ride-Problem (liDARP) wurde von Reiter, Schmidt und Stiglmayr [RSS24] als eigenständiges Optimierungsproblem eingeführt. Es verbindet die räumliche Struktur eines klassischen Linienbusses mit der zeitlichen Flexibilität von On-Demand-Ridepooling und schließt damit konzeptionell eine Lücke zwischen diesen beiden Betriebsformen. In der Praxis finden sich konzeptionell verwandte Systeme: Das französische FLEX'HOP 72 sowie die deutschen Systeme NAHBUS und der Rufbus Eiderstedt sind als reale Vorbilder für das liDARP angeführt [RSS24]. Allgemeiner ist in der deutschen Verkehrsliteratur das Konzept des *Bedarfslinienbetriebs* etabliert, das jedoch typischerweise einen Fahrplan beibehält. Das liDARP ergänzt diesen Ansatz um zeitliche Flexibilität und ist in dieser strikten Form bislang nicht in der Praxis umgesetzt [LMR⁺26].

Im Kern operieren im liDARP Fahrzeuge auf einer vordefinierten Linie, die durch eine feste, geordnete Folge von Haltestellen beschrieben wird. Anders als im klassischen Linienbetrieb müssen Fahrzeuge nicht alle Haltestellen anfahren: Sie können Haltestellen überspringen, an Haltestellen warten und die Fahrtrichtung wechseln. Das entscheidende Strukturmerkmal, das das liDARP vom allgemeinen DARP unterscheidet, ist die *Direktionalitätseigenschaft*: Gewendet wird nur ohne Fahrgäste an Bord (formal in Abschnitt 2.4).

Das Betriebsmodell lässt sich in den breiteren Kontext semi-flexibler Transportsysteme einordnen [LMR⁺26]: Klassische Liniensysteme sind sowohl räumlich als auch zeitlich starr; vollständig On-Demand-Systeme sind in beiden Dimensionen flexibel. Das liDARP ist räumlich durch die Linie gebunden, zeitlich aber flexibel. Im Vergleich zum allgemeinen DARP ist das liDARP durch die Linienbeschränkung rechnerisch günstiger strukturiert, da die Reihenfolge möglicher Abholpunkte durch die Liniengeographie vorgegeben ist [RSS24]. Dennoch ist auch das liDARP unter den hier betrachteten Annahmen (insbesondere bei Vorhandensein von Zeitfenstern) stark NP-schwer [LRS25], weshalb exakte Verfahren nur bis zu moderaten Instanzgrößen praktikabel sind.

Die Zielsetzung im liDARP ist typischerweise zweidimensional: Die Anzahl der bedienten Fahrgäste soll maximiert und gleichzeitig die von den Fahrzeugen tatsächlich zurückgelegte Strecke minimiert werden, um Emissionen gegenüber dem Individualverkehr einzusparen [RSS24]. Zudem werden zwei Serviceversprechen gegenüber bedienten Fahrgästen eingehalten: eine Obergrenze für die Fahrtzeit (als Vielfaches der Direktfahrtzeit) sowie ein Zeitfenster für den Abholzeitpunkt [RSS24]; beides wird in Abschnitt 2.4 formalisiert.

Seit der Einführung des liDARP wurden mehrere Erweiterungen untersucht. Barth, Reiter und Schmidt [BRS25] führen das liDARP mit Umstiegen (liDARPT) ein, bei dem Fahrgäste an Transferstationen zwischen verschiedenen Linien wechseln können, eine realitätsnähere Modellierung für verzweigte Liniennetze. Lauerbach et al. [LMR⁺26] betrachten eine Variante ohne Zeitfenster und entwickeln dafür sowohl einen exakten Branch-and-Price-Algorithmus, der Instanzen mit bis zu 55 Anfragen innerhalb von 60 Minuten mit einem MIP-Gap unter 5 % löst, als auch eine darauf aufbauende Root-Node-Heuristik, die auf Instanzen mit bis zu 100 Anfragen innerhalb von 15 Minuten Optimalitätslücken unter 5 % erreicht. Die mathematische Definition des in dieser Arbeit betrachteten statischen liDARP folgt im nächsten Abschnitt.

2.4 Mathematische Definition des statischen liDARP

Das statische linienbasierte Dial-a-Ride-Problem ist wie folgt definiert [RSS24].

Linie und Fahrzeuge. Die Linie wird durch eine feste, geordnete Folge von n Haltestellen $H = (1, \dots, n)$ beschrieben. Eine homogene Flotte von κ Fahrzeugen mit einheitlicher Kapazität $Q_{\max}$ bedient diese Linie; Fahrzeuge können ihre Routen an beliebigen Haltestellen beginnen und beenden. Zwischen je zwei Haltestellen $i, j \in H$ ist eine (Zeit-)Distanz $t_{i,j} \geq 0$ gegeben, die die Fahrzeit von i nach j modelliert und die Dreiecksungleichung erfüllt. Der Zeitbedarf für ein Wendemanöver ist an allen Haltestellen gleich und beträgt t_{turn} ; formal gilt $t_{i,i} := t_{\text{turn}}$ für alle $i \in H$.

Anfragen. Eine Menge R von m Anfragen ist vor Planungsbeginn vollständig und sicher bekannt (statisch-deterministischer Fall). Jede Anfrage $r \in R$ ist durch ein Tupel $(o_r, d_r, [e_r, l_r], q_r, b_r)$ beschrieben, wobei

- $o_r \in H$ die Abholhaltestelle (Origin),
- $d_r \in H$ die Zielhaltestelle (Destination) mit $o_r \neq d_r$,
- $e_r \geq 0$ der früheste Abfahrtszeitpunkt am Abholpunkt und $l_r \geq 0$ der späteste Ankunftszeitpunkt am Zielpunkt,
- $q_r \in \mathbb{N}$ die Anzahl der Fahrgäste (Last der Anfrage) und
- $b_r \geq 0$ die Bedienzeit für Ein- und Aussteigevorgänge

bezeichnen. Ein- und Aussteigen an derselben Haltestelle erfolgen aufeinanderfolgend: Wenn die Zielhaltestelle einer Anfrage gleichzeitig der Abholpunkt einer anderen Anfrage

ist und beide vom selben Fahrzeug bedient werden, muss der erste Fahrgast ausgestiegen sein, bevor der zweite einsteigt [RSS24]. Fahrgäste wechseln nicht zwischen Fahrzeugen.

Direktionalitätseigenschaft. Fahrzeuge dürfen Haltestellen überspringen (Abkürzungen nehmen), an Haltestellen warten und die Fahrtrichtung wechseln. Ein Wendemanöver ist jedoch *ausschließlich zulässig, wenn kein Fahrgast an Bord ist*. Diese *Direktionalitätseigenschaft* (*directionality property*) garantiert, dass jeder Fahrgast zu jedem Zeitpunkt in Richtung seines Ziels befördert wird [RSS24, LRS25]. Formal werden Anfragen mit $o_r < d_r$ als *aufsteigende Anfragen* und solche mit $o_r > d_r$ als *absteigende Anfragen* bezeichnet; aufsteigende Anfragen werden ausschließlich auf aufsteigenden Fahrabschnitten bedient, absteigende auf absteigenden.

Serviceversprechen. Gegenüber jedem bedienten Fahrgast werden zwei Serviceversprechen eingehalten [RSS24]:

1. *Maximale Fahrtzeit:* Die tatsächliche Fahrtzeit L_r darf das α -fache der Direktfahrtzeit nicht überschreiten:

$$L_r \leq L_r^{\max} := \alpha \cdot t_{o_r, d_r}, \quad \alpha \geq 1.$$

2. *Zeitfenstereinhaltung:* $\beta \geq 0$ ist die maximale zeitliche Abweichung von den Wunschzeitpunkten. Der tatsächliche Abholzeitpunkt darf höchstens β später liegen als der früheste Abfahrtszeitpunkt e_r , der tatsächliche Ankunftszeitpunkt höchstens β früher als der späteste Ankunftszeitpunkt l_r . Es gelten also gleichzeitig

$$\text{Pickup-Zeit} \in [e_r, e_r + \beta] \quad \text{und} \quad \text{Dropoff-Zeit} \in [l_r - \beta, l_r].$$

Zielfunktion. Das Optimierungsproblem entscheidet, welche Anfragen bedient und welche abgelehnt werden, wie bediente Anfragen den Fahrzeugen zugeordnet und wie die Fahrzeugrouten (also die Folgen besuchter Haltestellen mit den zugehörigen Ankunfts- und Abfahrtszeiten) geplant werden. Die Zielfunktion maximiert eine gewichtete Kombination zweier Komponenten [RSS24]:

1. die Anzahl der *bedienten Fahrgäste* und
2. die *eingesparte Distanz*: die Differenz zwischen der Summe $\sum_{r \in R^{\text{bed}}} t_{o_r, d_r}$ der direkten Distanzen über die Menge $R^{\text{bed}} \subseteq R$ der bedienten Anfragen (in der Terminologie von Liebchen et al. [LLMS21] die *Buchungskilometer ohne Umwege*) und der von den Fahrzeugen tatsächlich zurückgelegten Gesamtstrecke (*Fahrzeugkilometer gesamt*).

Die eingesparte Distanz quantifiziert den Umweltvorteil des Ridepoolings gegenüber individuellen Pkw-Fahrten.

Sublinienstruktur. Die Direktionalitätseigenschaft impliziert eine strukturell wichtige Eigenschaft: Die Route jedes Fahrzeugs lässt sich in eine Folge von *Sublinien* zerlegen [RSS24]. Eine Sublinie ist ein maximal langer Teilabschnitt der Route, auf dem

das Fahrzeug ohne Wendemanöver in einer Richtung fährt. Sublinien sind entweder *aufsteigend* (Haltestellen $i \rightarrow j$ mit $i < j$) oder *absteigend* ($i > j$); jede bediente Anfrage wird genau einer Sublinie zugeordnet. Diese Struktur bildet die Grundlage der von Reiter et al. [RSS24] entwickelten MILP-Formulierungen, die als Referenzverfahren in Abschnitt 3.1 beschrieben werden.

3 Verwandte Arbeiten

3.1 Exakte Lösungsverfahren und Komplexität des liDARP

Reiter, Schmidt und Stiglmayr [RSS24] entwickeln drei MILP-Formulierungen für das statische liDARP, die alle die Sublinienzerlegung der Fahrzeugrouten ausnutzen. Praktisch relevant für exakte Lösungen ist dabei vor allem die ereignisbasierte Formulierung (*event-based formulation*): Sie löst alle publizierten Benchmark-Instanzen (bis zu 58 Anfragen) in unter einer Sekunde, während die beiden anderen Formulierungen bei größeren Instanzen den 60-Minuten-Solver-Timeout erreichen [RSS24].

Die *ereignisbasierte Formulierung* adaptiert den von Gaul et al. [GKS22] für Ridepooling-Anwendungen vorgeschlagenen Ereignisgraphen auf das liDARP: Knoten kodieren $Q_{\max}$ -Tupel zulässiger Fahrzeugbelegungen, deren erster Eintrag die zuletzt ausgeführte Aktion (Pickup oder Dropoff, also Zu- oder Ausstieg) angibt; Kanten verbinden direktionalitäts- und zeitlich kompatible Ereignisse. Durch diese kompakte Kodierung kann der Solver große Teile des Suchraums implizit ausschließen, der entscheidende Grund für die oben genannte Leistungsstärke.

Die beiden anderen Formulierungen nutzen die Sublinienzerlegung auf unterschiedliche Weise: Die sublinienbasierte Formulierung ist eine eigenständige neue Modellierung des liDARP, die ortsbasierte überträgt ein bestehendes DARP-Modell auf die Linienstruktur. Beide sind in der Laufzeit aber langsamer als die ereignisbasierte Variante. Die *sublinienbasierte Formulierung* weist Anfragen direkt Sublinien zu und modelliert damit die Problemstruktur explizit, erzeugt aber so viele Variablen, dass für $Q_{\max} = 3$ mit Ausnahme der kleinsten Instanz mit zwei Fahrzeugen und 16 Anfragen (w2-16) alle Benchmark-Instanzen den 60-Minuten-Timeout erreichen. Die Instanznamen geben durchgehend die Fahrzeuganzahl (nach dem w) und die Anfragezahl (nach dem Bindestrich) an. Die *ortsbasierte Formulierung* adaptiert das klassische 3-Index-Modell des DARP [CL07] auf die Linienstruktur, indem aufsteigende und absteigende Haltestellen als getrennte Knotenmengen modelliert und durch Start-Turn- und End-Turn-Knoten verbunden werden. Sie skaliert deutlich besser als die sublinienbasierte Variante, erreicht aber bei den zwei größten Benchmark-Instanzen (w4-58 und w5-50) ebenfalls die Zeitgrenze.

Lauerbach, Reiter und Schmidt [LRS25] klassifizieren die Komplexität des liDARP detailliert und bestätigen, dass das Problem unter den hier betrachteten Bedingungen (insbesondere bei Vorhandensein von Zeitfenstern) stark NP-schwer bleibt.

Für deutlich größere Instanzen (etwa für Betriebsszenarien mit mehreren hundert Anfragen) liegt bislang keine systematische Evaluation der exakten Verfahren vor; wie sie in dieser Größenordnung skalieren, ist offen. Heuristische Verfahren bilden in diesem Größenbereich daher eine sinnvolle Ergänzung zu den exakten Ansätzen. Über die MILP-

Formulierungen von Reiter et al. hinaus liefern Lauerbach et al. [LMR⁺26] für die Variante ohne Zeitfenster einen Branch-and-Price-Algorithmus. Für die zeitfensterbehaftete Variante dieser Arbeit ist kein vergleichbarer Dekompositionsansatz verfügbar.

3.2 Tabu-Suche für das dynamische linienbasierte DARP

Heins Tabu-Suche [Hei24] wird hier aus zwei Gründen ausführlicher behandelt. Erstens ist sie zu dem in dieser Arbeit entwickelten Verfahren *methodisch eng verwandt*: Ihre Insert-, Delete- und Move-Operationen auf Sublinien sind strukturell analog zu den Einfügeoperationen, auf denen Greedy- und Regret-Insertion aufbauen. Zweitens liefert sie einen *empirischen Anhaltspunkt* dafür, welche Akzeptanzraten im liDARP erreichbar sind, und damit einen Bezugswert, an dem sich die Ergebnisse dieser Arbeit einordnen lassen. Der Abschnitt stellt das Verfahren daher kompakt vor und arbeitet anschließend die Unterschiede zum statischen liDARP heraus.

Das linienbasierte DARP tritt nicht nur in seiner statischen Variante auf, bei der alle Anfragen vor Planungsbeginn bekannt sind, sondern auch in dynamischen Varianten, in denen Anfragen erst während des laufenden Betriebs eintreffen. Hein [Hei24] entwickelt eine Tabu-Suche für das *zweiphasige linienbasierte dynamische DARP* (2P-LiDyDARP). Dieses ist im Kern dasselbe Problem wie das hier betrachtete statische liDARP (es teilt Linienstruktur, Direktionalitätseigenschaft und Sublinienzerlegung) und unterscheidet sich nur durch die dynamische Anfrageankunft. Es ist zudem im gleichen Forschungsumfeld entstanden. Der entscheidende Unterschied liegt in der Anfragedynamik: Während das statische liDARP alle Anfragen bei Planungsbeginn kennt, kommen im 2P-LiDyDARP Anfragen zu unterschiedlichen Zeitpunkten an. Fahrzeuge und Fahrgäste operieren simultan, Planungsentscheidungen müssen in Echtzeit und ohne vollständige Vorabinformation getroffen werden.

Das 2P-LiDyDARP gliedert sich in zwei Phasen. In *Phase 1* sind noch keine Fahrzeuge im Einsatz, Fahrgäste können aber bereits Anfragen einreichen. Am Übergang zu Phase 2 plant die gewählte Transportstrategie für alle bis dahin eingegangenen Anfragen in einem einzigen Schritt die Aktionspläne der Fahrzeuge. In *Phase 2* operieren Fahrzeuge und Fahrgäste gleichzeitig; neue Anfragen müssen einen Mindestabstand zwischen Anfragezeitpunkt und gewünschtem Abfahrtszeitpunkt (das *MinAnfragefenster*) einhalten, während Phase-1-Anfragen dieser Einschränkung nicht unterliegen. In jedem Planungszyklus der Phase 2 wird die Transportstrategie auf den aktuellen Zustand angewendet, um neu eingegangene Anfragen einzuplanen und bestehende Pläne anzupassen.

Die strukturellen Rahmenbedingungen sind weitgehend mit denen des statischen liDARP identisch: eine feste geordnete Linie, eine homogene Fahrzeugflotte mit einheitlicher Kapazität sowie die Direktionalitätseigenschaft, die Wendemanöver ausschließlich im Leerzustand erlaubt. Abweichend vom statischen liDARP, das zwei unabhängige Zeitfenster vorsieht, gibt das 2P-LiDyDARP nur eine maximale Wartezeit $W_{\max}$ am Abholpunkt vor; aus ihr leitet sich die späteste Ankunftszeit $r^l := r^g + W_{\max} + 2 \cdot T_{r^o, r^d}$ ab [Hei24]. Die Zielsetzung kombiniert Fahrgast- und Betriebsinteressen: Das 2P-LiDyDARP minimiert eine gewichtete Kombination aus mittlerer Wartezeit, mittlerer

Transportzeit und Gesamtfahrzeit der Flotte, mit starker Gewichtung der fahrgastorientierten Komponenten.

Hein vergleicht zwei Transportstrategien, die in beiden Phasen konsistent angewendet werden [Hei24]. Die *Greedy-Strategie* fügt jede neue Anfrage mittels einer Insert-Operation in alle in Frage kommenden Fahrzeuge ein und wählt das Ergebnis mit dem niedrigsten Wert der weiter unten definierten Bewertungsfunktion $f(s)$ (hier allein nach mittlerer Wartezeit gewichtet, also $\alpha = 100$ und $\beta = \gamma = 0$); kann die Anfrage in kein Fahrzeug eingefügt werden, wird sie abgelehnt. Die *Tabu-Search-Strategie* setzt auf der Greedy-Strategie auf: Sie führt zunächst denselben Greedy-Schritt aus und startet anschließend (sofern seit der letzten Tabu-Search mindestens 50 Simulationsticks, etwa fünf Minuten, vergangen sind) eine Tabu-Search zur Verbesserung der erhaltenen Lösung; deren Bewertungsfunktion gewichtet alle drei Komponenten mit $\alpha = \beta = 100$ und $\gamma = 1$, womit Wartezeit und Transportzeit gegenüber der Gesamtfahrzeit der Flotte stark priorisiert werden [Hei24].

Die Tabu-Search stützt sich auf drei Grundoperationen [Hei24]. Die *Insert-Operation* integriert eine Anfrage in das Aktionsprotokoll eines Fahrzeugs; je nach Lage der Start- und Zielhaltestelle relativ zu bestehenden Sublinien wird eine bestehende Sublinie abgeändert oder eine neue Fahrt geplant. Die *Delete-Operation* entfernt eine Anfrage aus dem Aktionsprotokoll, wobei vier strukturelle Fälle zu unterscheiden sind, die bestimmen, wie die verbleibenden Aktionen angepasst werden. Die *Move-Operation* kombiniert Delete und Insert: Sie entfernt eine Anfrage aus der Route eines Fahrzeugs und fügt sie in die Route eines anderen ein. Move-Operationen bilden die Nachbarschaftsstruktur der Tabu-Search; Insert und Delete dienen dabei als interne Hilfsoperatoren.

In jeder Iteration der Tabu-Search wird die Nachbarschaft $N(s)$ der aktuellen Lösung s gebildet, indem für jede aktuell veränderbare Anfrage geprüft wird, in welches andere Fahrzeug sie verschoben werden kann. Aus den gültigen Nachbarn wählt die Suche denjenigen mit dem niedrigsten Wert der Bewertungsfunktion

$$f(s) = \alpha \cdot \bar{w} + \beta \cdot \bar{t} + \gamma \cdot T_{\text{ges}},$$

wobei $\bar{w}$ die mittlere Wartezeit, $\bar{t}$ die mittlere Transportzeit und T_{ges} die Gesamtfahrzeit bezeichnet. Der ausgeführte Zug (die verschobene Anfrage zusammen mit dem Fahrzeug, in das sie eingefügt wurde) wird anschließend für eine festgelegte Anzahl von Iterationen (*TabuZeit*) auf die Tabuliste gesetzt und ist in diesem Zeitraum gesperrt. Die Suche terminiert, sobald entweder eine maximale Iterationszahl (*TabuIterationen*) erreicht ist oder über *MaximalKeineÄnderung* aufeinanderfolgende Iterationen keine neue beste Lösung gefunden wurde [Hei24].

In experimentellen Auswertungen auf synthetischen Instanzen vergleicht Hein die Tabu-Suche gegen eine Greedy-Baseline und eine weitere veröffentlichte Vergleichsstrategie [Hei24]. Die Akzeptanzrate der Tabu-Suche liegt bei 400 Anfragen bei 0,60, während die Greedy-Baseline 0,55 und die weitere Vergleichsstrategie 0,45 erreicht, eine relative Verbesserung von bis zu 33 % gegenüber dem schwächsten Referenzverfahren (eigene Berechnung aus den Ergebnissen von [Hei24]). Die mittlere Wartezeit verbessert sich auf 67 Ticks bei 400 Anfragen gegenüber 74 Ticks beim Greedy-Ansatz. Die Fahrzeugauslastung steigt auf 0,43 gegenüber 0,39. Der Rechenaufwand bleibt dabei praktikabel und

liegt pro Planungsschritt stets unter einer Sekunde, was den Einsatz im dynamischen Betriebskontext ermöglicht.

Im Vergleich zu dem in dieser Arbeit entwickelten Verfahren bestehen drei wesentliche Unterschiede. Erstens ist Heins Verfahren ein *verbesserungsbasiertes* Metaheuristikum: Es startet aus einer Greedy-Initiallösung und verbessert diese durch wiederholte lokale Umstrukturierungen. Die Qualität des Gesamtverfahrens hängt damit von beiden Komponenten ab. Die Regret- k -Insertion dieser Arbeit ist hingegen ein *konstruktives* Verfahren, das eine Lösung in einem einzigen Durchlauf aufbaut, ohne nachgelagerte Optimierung. Statt eine bereits gefundene Lösung zu reparieren, trifft Regret- k in jedem Schritt eine informiertere Einfügeentscheidung, indem sie Opportunitätskosten berücksichtigt: Die Anfrage, deren Nichtberücksichtigung den größten späteren Verlust bedeuten würde, wird priorisiert [RP06]. Zweitens adressiert Hein das dynamische Szenario, in dem Anfragen erst während des Betriebs eintreffen. Es liegt also keine vollständige Vorabinformation vor, und die Rechenzeit pro Planungsschritt ist knapp. Die statische Variante dieser Arbeit kennt dagegen alle Anfragen im Voraus und erlaubt eine tiefere Vorausplanung. Drittens unterscheiden sich die Zielfunktionen: Heins Verfahren minimiert eine gewichtete Kombination aus Wartezeit, Transportzeit und Gesamtfahrzeit. Diese Arbeit maximiert eine gewichtete Kombination aus bedienten Anfragen und eingesparter Distanz. Die eingangs genannte methodische Verwandtschaft bleibt damit trotz aller Unterschiede bestehen: Heins berichtete Akzeptanzraten dienen in dieser Arbeit als grober Vergleichsanker für die im liDARP erreichbare Lösungsqualität.

3.3 Regret-Heuristiken in der Vehicle-Routing-Literatur

Das Konzept der Regret-Insertion entstammt dem Bereich der konstruktiven Heuristiken für kombinatorische Optimierungsprobleme. *Regret-Insertion* bezeichnet dabei das allgemeine Prinzip, die *Regret- k -Insertion* die parametrische Familie, in der der Parameter k festlegt, wie viele Einfügealternativen in den Regret-Wert eingehen. Die Grundidee, nicht die Anfrage mit dem günstigsten Einfügepunkt zu priorisieren, sondern jene, deren Aufschieben den größten späteren Schaden anrichten würde, lässt sich als formale Opportunitätskostenrechnung in Insertion-Verfahren integrieren. Ropke und Pisinger [RP06] verwenden Regret-Insertion als einen von mehreren Insertion-Operatoren innerhalb einer Adaptive Large Neighborhood Search (ALNS) für das Pickup-and-Delivery-Problem mit Zeitfenstern (PDPTW). Sie führen dabei eine parametrische Regret- k -Familie ein, bei der k bestimmt, wie viele alternative Fahrzeuge in die Berechnung des Regret-Werts eingehen. Für das Dial-a-Ride-Problem speziell liefern Diana und Dessouky [DD04] das methodisch nächstliegende Verfahren zu dieser Arbeit: eine *parallele* Regret-Insertion-Heuristik für großskalige DARP-Instanzen mit Zeitfenstern. Der Zusatz *parallel* besagt, dass in jedem Schritt eine Anfrage über alle Fahrzeugrouten gleichzeitig bewertet und an ihrer besten Stelle eingefügt wird, anders als bei sequentiellen Verfahren, die eine Route nach der anderen vollständig befüllen.

Diana und Dessouky [DD04] betrachten ein Multi-Vehicle-DARP mit Zeitfenstern und servicebezogenen Qualitätsbeschränkungen, das den Anforderungen von Paratransit-Diensten in Los Angeles County nachempfunden ist. Jede Anfrage gibt entweder den frühesten Abholzeitpunkt oder den spätesten Ablieferzeitpunkt vor; daraus und aus einer maximal zulässigen Fahrtzeit (als Vielfaches der direkten Fahrtzeit) leiten sich die übrigen Zeitfenster ab. Die Zielfunktion minimiert eine gewichtete Kombination aus Gesamtfahrstrecke aller Fahrzeuge, summierter Mehrfahrtzeit gegenüber der Direktfahrt und GesamtLeerlaufzeit der Flotte; wobei Mehrfahrtzeit am stärksten gewichtet wird, um die Fahrgastperspektive zu priorisieren. Anders als im liDARP dieser Arbeit werden dabei alle Anfragen bedient: Das Verfahren lehnt keine Anfrage ab, sondern minimiert Fahrzeugzahl und Strecke; eine Akzeptanzrate als Zielgröße existiert dort nicht. Dieser Unterschied ist beim späteren Vergleich der Akzeptanzraten zu beachten. Getestet wird das Verfahren auf synthetischen Instanzen mit 500 und 1000 Anfragen, die auf Basis echter Nachfrageverteilungen aus dem Los Angeles County generiert wurden.

Der Algorithmus gliedert sich in zwei Phasen. In der ersten Phase werden Startanfragen (*Seed Requests*) für jede Fahrzeugroute ausgewählt. Neben dem frühesten Abholzeitpunkt berücksichtigt die Auswahl einen *Dezentralisierungsindex* D_k , der die räumliche Isolation einer Anfrage relativ zur Gesamtanfrageverteilung misst: Je isolierter eine Anfrage liegt, desto größer ist D_k und desto eher wird sie als Startanfrage priorisiert [DD04]. Ein Schwellenwert $a \in [0, 1]$ entscheidet, ob eine Anfrage primär nach räumlicher Isolation oder nach frühestem Abholzeitpunkt als Startanfrage ausgewählt wird. Diese Initialisierung verhindert den bei rein zeitbasierter Auswahl auftretenden Effekt, dass räumlich abgelegene Anfragen erst spät eingeplant werden; zu einem Zeitpunkt, an dem kaum noch Einfügemöglichkeiten verbleiben.

In der zweiten Phase werden die verbleibenden Anfragen parallel mittels Regret-

Insertion in die bestehenden Routen eingefügt. In der Formulierung von Ropke und Pisinger [RP06] wird für jede noch nicht eingeplante Anfrage i in jedem Fahrzeug die beste zulässige Einfügeposition bestimmt; jedes Fahrzeug trägt damit einen Bestwert bei. Diese Fahrzeug-Bestwerte werden aufsteigend sortiert:

$$c_i^{(1)} \leq c_i^{(2)} \leq \dots \leq c_i^{(\kappa)},$$

wobei $c_i^{(j)}$ die Kosten des j -besten Fahrzeugs für Anfrage i bezeichnen. Der *Regret-Wert* ist die Differenz zwischen dem besten und dem k -besten Fahrzeug:

$$\text{regret}_k(i) = c_i^{(k)} - c_i^{(1)}.$$

Da pro Fahrzeug nur ein Bestwert beiträgt, ist k höchstens so groß wie die Fahrzeuganzahl κ . Ropke und Pisinger [RP06] definieren für allgemeines k eine Summenform $\sum_{j=1}^k (c_i^{(j)} - c_i^{(1)})$. Die in dieser Arbeit verwendete Zwei-Term-Variante $c_i^{(k)} - c_i^{(1)}$ stimmt für $k = 2$ exakt damit überein und ist für $k \geq 3$ eine in der Literatur verbreitete Vereinfachung. Diana und Dessouky [DD04] übernehmen die Summenform über alle Routen, was in der Notation von Ropke und Pisinger einer *Regret- m -Heuristik* entspricht (m = Fahrzeuganzahl). Fahrzeuge, in denen für i keine zulässige Einfügeposition existiert, erhalten einen sehr großen Bestwert. Die Einfügekosten messen die Änderung des Zielfunktionswerts, die das Einreihen von i an einer bestimmten Position verursacht: zusätzlich gefahrene Strecke, veränderte Leerlaufzeiten und erhöhte Mehrfahrzeiten anderer Fahrgäste fließen gewichtet ein. In jedem Schritt wird die Anfrage mit dem höchsten Regret-Wert als nächste eingeplant; an der besten Position im besten Fahrzeug ($c_i^{(1)}$). Der Regret-Wert quantifiziert damit den Verlust, den das Aufschieben einer Anfrage verursacht: Eine Anfrage mit hohem Regret hat wenige Fahrzeuge, die sie zu vergleichbar guten Kosten aufnehmen können. Ihr frühzeitiger Einbau sichert das noch verfügbare beste Fahrzeug.

Der Parameter k bestimmt, wie viele Alternativen in die Bewertung eingehen; seine Rolle wird in Abschnitt 4.3.1 im Detail behandelt.

Die Überlegenheit der Regret-Insertion gegenüber Greedy-Insertion beruht auf zwei komplementären Effekten [DD04]. Erstens vermeidet sie *myopische Entscheidungen*: Statt wie ein Greedy-Verfahren stets die aktuell billigste Anfrage zu nehmen, priorisiert sie jene Anfragen, die bei Aufschub am stärksten benachteiligt wären. Zweitens wirkt der Regret-Wert als *Machbarkeitsfilter*: Anfragen, für die nur wenige Fahrzeuge eine zulässige Einfügung erlauben, erhalten hohe Regret-Werte, weil der Kostensprung zwischen dem besten und dem k -besten Fahrzeug durch die großen Strafwerte unzulässiger Fahrzeuge verstärkt wird. Diese Anfragen werden früh eingeplant, bevor die Zeitfenster- und Kapazitätsbeschränkungen durch spätere Einfügungen weiter verengt werden.

Experimentell erzielt die reine Regret-Insertion von Diana und Dessouky [DD04] (gemessen auf ihrem allgemeinen Multi-Vehicle-DARP, nicht auf dem liDARP) gegenüber paralleler Solomon-Insertion eine Reduktion der benötigten Fahrzeuganzahl um 6,25 % (1000 Anfragen) bzw. 7,39 % (500 Anfragen) und der Leerkilometer um rund 20 %. Die Übertragbarkeit dieser Größenordnungen auf das liDARP ist mangels vergleichbarer Stu-

dien offen. Kombiniert mit einer verbesserten Startanfragen-Auswahl verbessern sich diese Werte auf 8,2 % weniger Fahrzeuge und 24,6 % weniger Leerkilometer. Die Gesamtfahrstrecke und die Fahrgastfahrtzeiten verbessern sich um jeweils rund 6 %. In einer Sensitivitätsanalyse über Szenariotypen mit lockeren, mittleren und engen Zeitfenster- und Fahrzeitbeschränkungen bleibt die Überlegenheit des Regret-Verfahrens durchgängig erhalten. Sie fällt bei sehr engen Beschränkungen etwas geringer aus, da dort kaum zulässige Alternativen existieren und der Regret-Wert seinen Differenzierungsvorteil verliert. Die rechnerische Komplexität beträgt $\mathcal{O}(n^3)$ und liegt damit über den $\mathcal{O}(n^2)$ klassischer sequentieller Insertion-Heuristiken, aber deutlich unter dem Aufwand lokaler Suchverfahren [DD04]. Bei 1000 Anfragen liefert der Algorithmus Lösungen in rund 195 Minuten und bei 500 Anfragen in rund 26 Minuten.

Das in dieser Arbeit entwickelte Verfahren überträgt das Regret-Prinzip auf das statische liDARP. Der konzeptionelle Kern (Regret-Wert als Differenz zwischen k -ter und bester Einfügeoption, Priorisierung der Anfrage mit maximalem Regret) bleibt unverändert. Die wesentliche Anpassung betrifft die Berechnung der Einfügekosten: Während im allgemeinen DARP Anfragen an beliebigen Routenpositionen eingereiht werden können, sind im liDARP Einfügungen durch die Direktionalitätseigenschaft strukturell eingeschränkt. Aufsteigende Anfragen ($o_r < d_r$) können ausschließlich in aufsteigende Sublinien eingefügt werden und umgekehrt. Fahrzeugkapazität und Zeitfenster sind dabei (wie im allgemeinen DARP) ebenfalls einzuhalten; genuin neu im liDARP ist allein die durch die Direktionalitätseigenschaft erzwungene Bindung an die Sublinienrichtung. Dieser strukturelle Unterschied verändert, welche Positionen als zulässig gelten; und damit, welche Anfragen im liDARP als besonders schwierig einzuplanen gelten und daher hohe Regret-Werte erhalten. Die Direktionalität erzwingt außerdem eine enge Kopplung zwischen Anfragenrichtung und Sublinienstruktur, die im allgemeinen DARP kein Pendant hat und die Kostenberechnung pro Einfügeposition grundlegend verändert.

Methodisch ist die Regret- k -Insertion einer der zentralen Insertion-Operatoren in der ALNS von Ropke und Pisinger [RP06]. Ihre Anpassung an das liDARP, die diese Arbeit leistet, bildet damit einen Baustein, auf dem eine spätere vollständige ALNS für das liDARP aufsetzen kann.

4 Methodik

Diese Arbeit entwickelt eine Regret- k -Insertion für das liDARP; als Baseline dient eine Greedy-Insertion. Sie entspricht zugleich dem Spezialfall $k = 1$ der Regret- k -Familie, denn für $k = 1$ ergibt sich $\text{regret}_1(r) = 0$ für alle Anfragen, und das Verfahren wird zu Greedy. Beide sind *konstruktive Heuristiken*. Sie bauen eine Lösung durch schrittweises Einplanen von Anfragen auf, ohne eine bereits gefundene Lösung nachträglich zu modifizieren. Im Gegensatz zu Verbesserungsheuristiken wie ALNS oder Tabu-Suche (vgl. Kapitel 3) werden bestehende Lösungen nicht durch Umsortieren oder Neuzuweisung nachbearbeitet. Abschnitt 4.1 legt das gemeinsame Grundgerüst beider Verfahren fest; Abschnitte 4.2 und 4.3 beschreiben Greedy- bzw. Regret-Insertion im Detail; Abschnitt 4.4 stellt beide Algorithmen in Pseudocodeform gegenüber und analysiert ihre Laufzeitkomplexität. Anfragen werden im Folgenden mit r bezeichnet. Dies entspricht dem in Kapitel 3 nach Ropke und Pisinger sowie Diana und Dessouky verwendeten Index i .

4.1 Gemeinsames Grundgerüst der Insertion-Heuristiken

4.1.1 Lösungsrepräsentation

Eine Lösung weist jedem Fahrzeug $v \in \{1, \dots, \kappa\}$ eine *Route* zu, die als geordnete Folge von Sublinien dargestellt wird. Jede Sublinie s ist ein maximal langer, richtungskonsistenter Fahrtabschnitt im Sinne von Abschnitt 2.4. Sie speichert ihre Richtung (aufsteigend $\uparrow$ oder absteigend $\downarrow$) sowie die geordnete Folge der bedienten Anfragen mit Abfahrts- und Ankunftszeitpunkten an jedem Halt. Die besuchten Haltestellen ergeben sich aus den Start- und Zielhaltestellen dieser Anfragen (o_r, d_r): Eine Haltestelle wird genau dann angefahren, wenn dort ein Pickup oder Dropoff stattfindet. Die Fahrzeuglast zwischen je zwei Haltepunkten wird als Implementierungshilfsgröße mitgeführt. Zwischen zwei aufeinanderfolgenden Sublinien wendet das Fahrzeug leer. Die Wendezeit t_{turn} wird in die Zeitplanung der nachfolgenden Sublinie eingerechnet.

Beispiele. Drei kleine Lösungen auf einer Linie mit vier Haltestellen $H = (1, 2, 3, 4)$ und einem Fahrzeug v_1 veranschaulichen die Repräsentation; aufeinanderfolgende Sublinien werden durch $\parallel$ getrennt. Aufgeführt sind nur Haltestellen mit Pickup, Dropoff oder Wende; reine Zwischenhaltestellen werden durchfahren und nicht genannt.

1. *Eine Anfrage, eine Sublinie.* Bei einer einzelnen aufsteigenden Anfrage $r_1 : 1 \rightarrow 3$ besteht die resultierende Route aus einer einzigen Sublinie ohne Wende (Haltestelle 2 wird durchfahren):

$$v_1 : \uparrow 1 (\text{Pickup } r_1) \rightarrow 3 (\text{Dropoff } r_1).$$

2. *Zwei Anfragen, eine Wende.* Für $r_1 : 1 \rightarrow 3$ und $r_2 : 4 \rightarrow 2$ sind zwei Sublinien nötig, da die Anfragen in entgegengesetzten Richtungen liegen:

$$\begin{aligned} v_1 : & \uparrow 1 (\text{Pickup } r_1) \rightarrow 3 (\text{Dropoff } r_1) \rightarrow 4 \parallel \\ & \downarrow 4 (\text{Pickup } r_2) \rightarrow 2 (\text{Dropoff } r_2). \end{aligned}$$

Beim Wenden an Haltestelle 4 ist das Fahrzeug leer. So verlangt es die Direktionalitätseigenschaft.

3. *Zwei Anfragen, ein gemeinsamer Fahrabschnitt.* Bei $r_1 : 1 \rightarrow 4$ und $r_2 : 2 \rightarrow 3$ werden beide Anfragen in derselben aufsteigenden Sublinie bedient:

$$v_1 : \uparrow 1 (\text{Pickup } r_1) \rightarrow 2 (\text{Pickup } r_2) \rightarrow 3 (\text{Dropoff } r_2) \rightarrow 4 (\text{Dropoff } r_1).$$

Die Fahrzeuglast steigt nach dem zweiten Pickup auf 2 und fällt nach dem ersten Dropoff wieder auf 1.

Zu Beginn eines jeden Algorithmuslaufs sind alle Fahrzeugrouten leer und alle Anfragen ungeplant. Wir bezeichnen mit $U \subseteq R$ die Menge der noch ungeplanten Anfragen; zu Beginn gilt $U = R$. Das Verfahren terminiert, wenn $U = \emptyset$ ist oder wenn für keine verbleibende Anfrage in U eine zulässige Einfügeposition mehr existiert. Im zweiten Fall werden die verbleibenden Anfragen abgelehnt, sodass U am Ende nicht zwingend leer ist.

4.1.2 Zulässigkeitsbedingungen

Das Einfügen einer Anfrage r in eine Sublinie s an Position p ist genau dann *zulässig*, wenn alle folgenden Bedingungen erfüllt sind:

1. *Direktionalität:* Die Richtung der Anfrage stimmt mit der Richtung der Sublinie überein, das heißt $o_r < d_r$ bei aufsteigenden und $o_r > d_r$ bei absteigenden Sublinien.
2. *Kapazität:* An keinem Haltepunkt der Sublinie übersteigt die Last nach der Einfügung die Fahrzeugkapazität $Q_{\max}$.
3. *Zeitfenster:* Abholung und Ablieferung von r liegen in ihren (abgeleiteten) Zeitfenstern (die Abholung frühestens bei e_r , die Ablieferung spätestens bei l_r), und die Einfügung verletzt die Zeitfenster keiner bereits eingeplanten Anfrage in s .
4. *Maximale Fahrtzeit:* Die Fahrtzeit L_r von Anfrage r überschreitet nicht $L_r^{\max} = \alpha \cdot t_{o_r, d_r}$.
5. *β -Serviceversprechen:* Der tatsächliche Abholzeitpunkt liegt im Intervall $[e_r, e_r + \beta]$ (vgl. Abschnitt 2.4).

Die Zulässigkeit wird für jede konkrete Position p einzeln geprüft: Dieselbe Anfrage kann in einer Sublinie an einer Stelle zulässig und an einer anderen unzulässig sein, da p das Last- und Zeitprofil bestimmt. Treffen an einer Haltestelle Aus- und Einstieg zusammen, gilt zudem die Reihenfolge aus Abschnitt 2.4: Zuerst steigen die Fahrgäste der dort endenden Anfrage aus, danach die der neuen ein. Die Kapazitätsprüfung berücksichtigt diese Reihenfolge.

Zusätzlich zu Einfügungen in bestehende Sublinien können neue Sublinien eröffnet werden, sofern das Fahrzeug zum Wendezeitpunkt leer ist und die durch Wendezeit t_{turn} und Anfahrt zu o_r entstehende Zeitverzögerung mit den Zeitfenstern der betroffenen Anfragen vereinbar ist. Eine neue Sublinie ist dabei immer eine einzelne Sublinie, eingefügt am Anfang, am Ende oder zwischen zwei bestehenden Sublinien. Das Fahrzeug setzt jeweils leer zu ihr über (siehe Abschnitt 4.3.2).

4.1.3 Einfügekosten

Die Zielfunktion des liDARP kombiniert die Anzahl bedienter Fahrgäste mit der eingesparten Distanz. In Abschnitt 2.4 haben wir festgelegt, dass die eingesparte Distanz nur über die *bedienten* Anfragen summiert wird; nicht über alle Anfragen aus R . Diese Wahl bestimmt die Form der Einfügekosten.

Wird eine ungeplante Anfrage r an Position p eingeplant, wächst der Zielfunktionswert um zwei Beiträge:

- $w_1 \cdot q_r$ für die nun bedienten q_r Fahrgäste der Anfrage,
- $w_2 \cdot (t_{o_r, d_r} - \Delta_p(r))$ für die hinzukommende Direktstrecke t_{o_r, d_r} von r abzüglich des entstehenden Mehrwegs $\Delta_p(r) \geq 0$.

Die Zielfunktion gewichtet hier die *Fahrgäste* q_r einer Anfrage. Die in der Evaluation verwendete Akzeptanzrate zählt dagegen *Anfragen*. Beide Maße fallen zusammen, sobald jede Anfrage genau einen Fahrgast umfasst ($q_r = 1$); andernfalls ist die Unterscheidung bei der Interpretation der Ergebnisse zu beachten.

Die *Einfügekosten* $c_p(r)$ sind das Negative dieser Summe:

$$c_p(r) = -(w_1 \cdot q_r + w_2 \cdot (t_{o_r, d_r} - \Delta_p(r))).$$

Kleinere Werte bedeuten einen größeren Zielfunktionszuwachs. Für unzulässige Positionen wird $c_p(r) = +\infty$ gesetzt.

Lokale Kosten, globale Zulässigkeit. Der Mehrweg $\Delta_p(r)$ erfasst nur die unmittelbar an der Einfügestelle zusätzlich gefahrene Strecke. Das Einfügen verschiebt jedoch alle nachfolgenden Ankunfts- und Abfahrtszeiten; innerhalb derselben Sublinie und, über deren veränderte Endzeit, auch in allen folgenden Sublinien desselben Fahrzeugs. Eine bereits eingeplante Anfrage mit engem Zeitfenster kann dadurch ihre Schranke überschreiten. Die Einfügekosten bleiben deshalb eine rein lokale Größe. Die *Zulässigkeit* einer Einfügung wird hingegen global geprüft, indem nach der Einfügung die gesamte

Zeitkette des Fahrzeugs fortgeschrieben und gegen die Zeitfenster aller betroffenen Anfragen abgeglichen wird. Verletzt die Verschiebung irgendeine Schranke, gilt die Position als unzulässig ($c_p(r) = +\infty$).

Hätten wir uns in Abschnitt 2.4 für die andere Variante entschieden (Summe über alle Anfragen aus R , auch die abgelehnten), so wäre t_{o_r, d_r} Teil einer konstanten Summe und ginge nicht in die Einfügekosten ein; nur der Mehrweg $\Delta_p(r)$ wäre relevant. Unsere Wahl belohnt also explizit, wenn lange Anfragen bedient werden, weil ihre Direktanzahl direkt in den Kostenausdruck eingeht.

4.2 Greedy-Insertion als Baseline

Die Greedy-Insertion trifft in jedem Schritt die lokal günstigste Einfügeentscheidung. Sie dient als Baseline-Verfahren und motiviert durch ihre strukturelle Schwäche den Einsatz der Regret- k -Insertion.

Zu Beginn sind alle Anfragen ungeplant und alle Routen leer. Die erste Anfrage in einem Fahrzeug kann daher nur durch das Anlegen einer neuen Sublinie eingeplant werden. In jedem Schritt wird für jede ungeplante Anfrage r die günstigste zulässige Einfügeposition über alle Fahrzeuge v und alle Positionen p bestimmt:

$$c_r^{(1)} = \min_{v, p} c_p(r).$$

Die Zahl der je Anfrage geprüften Positionen p ergibt sich aus den Sublinien passender Richtung: Eine Sublinie mit n Halten bietet $O(n^2)$ mögliche (Pickup-, Dropoff-)Indexpaare mit Pickup vor Dropoff; hinzu kommen die Optionen, am Anfang, am Ende oder zwischen zwei Sublinien eine neue Sublinie anzulegen. Über alle κ Fahrzeuge summiert bestimmt diese Anzahl den Rechenaufwand pro Iteration. Anschließend wird die Anfrage r^* mit dem kleinsten Wert $c_{r^*}^{(1)}$ ausgewählt und an ihrer besten Position eingeplant. Haben mehrere Anfragen denselben Minimalwert, wird deterministisch nach aufsteigendem Anfrageindex aufgelöst, um die Reproduzierbarkeit der Experimente zu gewährleisten. Der Algorithmus terminiert, wenn $U = \emptyset$ (alle Anfragen eingeplant) oder wenn für keine Anfrage in U eine zulässige Einfügeposition existiert ($c_r^{(1)} = +\infty$ für alle $r \in U$); in letzterem Fall werden alle verbleibenden Anfragen abgelehnt.

Das zentrale Problem der Greedy-Insertion ist ihre *Myopie*: Sie priorisiert die Anfrage mit dem aktuell geringsten Einfügekosten, ohne zu berücksichtigen, wie diese Entscheidung die Einplanungsmöglichkeiten noch ungeplanter Anfragen beeinflusst. Eine Anfrage, die im aktuellen Schritt günstig einfügbar ist, kann die Sublinienstruktur so verändern, dass für eine spätere Anfrage mit engem Zeitfenster keine zulässige Position mehr verbleibt; obwohl bei einer anderen Einplanungsreihenfolge beide hätten bedient werden können. Dieses Verhalten kann zu unnötigen Ablehnungen und einer suboptimalen Akzeptanzrate führen.

4.3 Regret- k -Insertion

4.3.1 Grundprinzip und Regret-Wert

Die Regret- k -Insertion begegnet der Myopie der Greedy-Insertion durch explizite Berücksichtigung von Opportunitätskosten. Statt die Anfrage mit dem günstigsten Einfügepunkt zu wählen, priorisiert sie jene, bei der der Verlust durch Aufschub am größten wäre.

Für jede ungeplante Anfrage r wird in jedem Fahrzeug die beste zulässige Einfügeposition bestimmt; jedes Fahrzeug trägt damit einen Bestwert bei. Aus diesen κ Fahrzeug-Bestwerten werden zwei Größen abgelesen: $c_r^{(1)}$, die Kosten des besten Fahrzeugs, und $c_r^{(k)}$, die Kosten des k -besten Fahrzeugs. Ist die Anzahl der Fahrzeuge mit mindestens einer zulässigen Einfügeposition kleiner als k , so wird $c_r^{(k)} := +\infty$ gesetzt, sodass Anfragen mit weniger als k verfügbaren Fahrzeugen automatisch maximalen Regret erhalten. Der *Regret-Wert* von r ist definiert als

$$\text{regret}_k(r) = c_r^{(k)} - c_r^{(1)}.$$

Er misst, wie stark die Qualität von der besten zur k -zweitbesten Einfügeoption abfällt, also den Verlust durch Aufschub von r . In jedem Schritt wird die Anfrage r^* mit dem höchsten Regret-Wert ausgewählt und an ihrer besten Position $c_{r^*}^{(1)}$ eingeplant:

$$r^* = \arg \max_{r \in U, c_r^{(1)} < +\infty} \text{regret}_k(r).$$

Dieser Schritt wird iteriert, bis alle Anfragen eingeplant oder abgelehnt sind. Die Implementierung verschärft diese Auswahl noch: Sie plant nur Anfragen ein, deren beste Einfügung den Zielwert *erhöht*, und schiebt zulässige, aber wertsenkende Einfügungen auf (Abschnitt 5.1.7). Haben mehrere Anfragen denselben maximalen Regret-Wert, wird zunächst die Anfrage mit dem günstigeren besten Einfügewert $c_r^{(1)}$ bevorzugt; bleibt auch dieser gleich, entscheidet der aufsteigende Anfrageindex (deterministisch, analog zu Abschnitt 4.2).

Der Parameter k steuert die Tiefe der Alternativbetrachtung. Für $k = 1$ gilt $\text{regret}_1(r) = 0$ für alle Anfragen. Alle Anfragen erscheinen gleich dringlich, und das Verfahren degeneriert zur Greedy-Insertion. Für $k = 2$ bewertet der Regret-Wert den Abstand zwischen dem besten und zweitbesten Fahrzeug. Er identifiziert Anfragen, die außer ihrem besten Fahrzeug kaum akzeptable Alternativen haben. Für $k = 3$ vergleicht der Regret-Wert das beste mit dem drittbesten Fahrzeug. Hat eine Anfrage etwa die sortierten Fahrzeug-Bestwerte $c_r^{(1)} = 10$, $c_r^{(2)} = 12$ und $c_r^{(3)} = 80$, so ergibt sich $\text{regret}_3(r) = 70$; obwohl das zweitbeste Fahrzeug mit 12 kaum schlechter als das beste ist. Der Sprung auf 80 signalisiert, dass die Anfrage nach den ersten beiden Fahrzeugen kaum noch unterzubringen wäre. Bei $k = 2$ wäre der Regret-Wert lediglich 2 und die Anfrage würde nicht auffallen. Größere Werte von k berücksichtigen ein breiteres Spektrum von Alternativen. k ist nach oben durch die Fahrzeuganzahl κ beschränkt. Bei nur einem Fahrzeug wäre $\text{regret}_k = 0$, und die Heuristik würde zur Greedy-Insertion entarten. Ihr Mehrwert hängt von der Instanzdichte ab: in Instanzen mit moderater Dichte differenzieren sie schwierige Anfragen

feiner; in dichten Instanzen mit vielen unzulässigen Positionen können dagegen mehrere Anfragen denselben Regret-Wert ($+\infty$) erhalten und werden ununterscheidbar. Ropke und Pisinger [RP06] setzen $k \in \{2, 3, 4\}$ ein. Da $k \leq \kappa$ gelten muss, ist in einer Instanz effektiv nur $\min(k, \kappa)$ wirksam: Ein Wert wie $k = 4$ lässt sich erst ab vier Fahrzeugen testen, während Instanzen mit weniger Fahrzeugen nur die entsprechend kleineren k zulassen. Welche Werte von k pro Instanz untersucht werden können, hängt damit von deren Fahrzeugzahl ab.

4.3.2 Zulässige Einfügepositionen im liDARP

Im allgemeinen DARP kann eine Anfrage prinzipiell an jeder beliebigen Position einer Fahrzeugroute eingefügt werden, solange Kapazität und Zeitfenster eingehalten werden. Im liDARP schränkt die Direktionalitätseigenschaft den Suchraum zulässiger Einfügepositionen strukturell ein und erzeugt eine enge Kopplung zwischen Anfragenrichtung und Sublinienstruktur.

Für eine Anfrage r mit $o_r < d_r$ (aufsteigende Richtung) kommen ausschließlich aufsteigende Sublinien als Einplanungsort in Frage; entsprechend kommen für absteigende Anfragen ($o_r > d_r$) nur absteigende Sublinien in Betracht. Innerhalb einer Sublinie s der passenden Richtung sind die Stationen o_r und d_r durch die Anfrage fest vorgegeben; variabel ist nur, an welchen *Stellen* der bestehenden Haltefolge von s Pickup und Dropoff eingefügt werden. Geprüft werden daher alle Index-Paare (i_P, i_D) mit $i_P \leq i_D$ in der Haltefolge von s , die die richtungskonsistente Stationsreihenfolge wahren (Pickup nicht nach Dropoff). An jeder dieser Positionen wird die vollständige Zulässigkeitsprüfung (Kapazität, Zeitfenster, α, β) durchgeführt.

Existiert für ein Fahrzeug kein geeigneter Einfügeplatz in bestehenden Sublinien, wird das Anlegen einer neuen Sublinie geprüft. Eine neue Sublinie der Richtung von r kann das Fahrzeug nur dort aufnehmen, wo es leer ist. Eine neue Sublinie wird immer als *einzelne* besetzte Sublinie an der gewählten Stelle eingefügt, also am Anfang, am Ende oder zwischen zwei bestehenden Sublinien. Das leere Umsetzen zum Pickup und (im Inneren) weiter zur nächsten Sublinie ist dabei eine *Leerfahrt*, keine eigene Sublinie. Weil das Fahrzeug leer wendet, dürfen auch zwei gleichgerichtete Sublinien aufeinanderfolgen; eine strenge Abwechslung der Richtungen ist nicht erforderlich. Dabei ist sicherzustellen, dass das Wendemanöver vor o_r und nach d_r den Direktionalitätsbedingungen genügt. Die Fahrt zur Eröffnung einer neuen Sublinie (Anfahrt zu o_r , Bedienung von r , Rückfahrt zum Einstiegspunkt der nächsten Sublinie) geht vollständig in den Zusatzfahraufwand $\Delta_p(r)$ ein.

4.3.3 Einfügekosten und Zielfunktionsbeitrag

Beim Einplanen einer Anfrage treten vier Fälle auf, die sich im Zusatzfahraufwand $\Delta_p(r)$ unterscheiden:

1. Einfügung in eine bestehende Sublinie passender Richtung,
2. neue Sublinie am Anfang der Route,

3. neue Sublinie am Ende der Route,
4. neue Sublinie im Inneren der Route (zwischen zwei bestehenden Sublinien).

Die Fälle 2 und 3 sind Randfälle des allgemeinen Ausdrucks für neue Sublinien, bei denen Anfahrts- bzw. Rückfahrtterm entfallen.

Für eine Einfügung in eine bestehende Sublinie zwischen den Haltestellen h_{vor} und h_{nach} setzt sich der Zusatzfahraufwand zusammen aus

$$\begin{aligned}\Delta_p(r) &= t_{h_{\text{vor}}, o_r} + t_{o_r, d_r} + t_{d_r, h_{\text{nach}}} - t_{h_{\text{vor}}, h_{\text{nach}}} \\ &= \text{Anfahrt zum Pickup} + \text{Direktfahrt der Anfrage} \\ &\quad + \text{Weiterfahrt zum Folgehalt} - \text{ursprüngliche Direktverbindung.}\end{aligned}$$

Bei der Anlage einer neuen Sublinie zwischen zwei bestehenden Sublinien s_{prev} und s_{next} gilt:

$$\begin{aligned}\Delta_p(r) &= t_{s_{\text{prev}}^{\text{end}}, o_r} + t_{o_r, d_r} + t_{d_r, s_{\text{next}}^{\text{start}}} - t_{s_{\text{prev}}^{\text{end}}, s_{\text{next}}^{\text{start}}} \\ &= \text{Fahrt zum Pickup} + \text{Direktfahrt der Anfrage} \\ &\quad + \text{Fahrt zur nächsten Sublinie} - \text{bisheriger Übergang.}\end{aligned}$$

Dabei bezeichnen $s_{\text{prev}}^{\text{end}}$ und $s_{\text{next}}^{\text{start}}$ die End- bzw. Startstation der angrenzenden Sublinien. Liegt die neue Sublinie am Anfang oder Ende der Route, entfallen die entsprechenden Anfahrts- bzw. Rückfahrtterme. Auch hier ist $\Delta_p(r)$ nur die lokale Kostengröße: Die durch die neue Sublinie verursachte Zeitverschiebung in die Folgesublinie und alle weiteren Sublinien wird, wie in Abschnitt 4.1.3 beschrieben, bei der Zulässigkeitsprüfung berücksichtigt.

In die Regret-Berechnung aus Abschnitt 4.3.1 geht von diesen Kosten nur die Differenz $c_r^{(k)} - c_r^{(1)}$ ein. Sie ist invariant gegenüber konstanten Verschiebungen des Kostenniveaus.

4.4 Algorithmische Gesamtübersicht

Greedy- und Regret- k -Insertion teilen dieselbe Schleifenstruktur. Der einzige Unterschied liegt in der Auswahlregel. Im Folgenden bezeichnet $\mathcal{S}$ die aktuelle Lösung (Menge der Fahrzeugrouten); U ist wie in Abschnitt 4.1.1 definiert.

Abb. 4.1: Greedy-Insertion in Pseudocode (Algorithmus 1).

```

 $U \leftarrow R; \quad \mathcal{S} \leftarrow \text{leere Routen}$ 
while  $U \neq \emptyset$  do
  for each  $r \in U$ :
    for each Fahrzeug  $v$ :  $c_v(r) \leftarrow \min_p c_p(r)$  (beste Position in  $v$ )
     $c_r^{(1)} \leftarrow \min_v c_v(r)$ 
    if alle  $c_r^{(1)} = +\infty$  then break
     $r^* \leftarrow \arg \min_{r \in U, c_r^{(1)} < +\infty} c_r^{(1)}$  (Ties: kleinster Index zuerst)
    füge  $r^*$  an bester Position in  $\mathcal{S}$  ein;  $U \leftarrow U \setminus \{r^*\}$ 
return  $\mathcal{S}, U$ 

```

Abb. 4.2: Regret- k -Insertion in Pseudocode (Algorithmus 2).

```

 $U \leftarrow R; \quad \mathcal{S} \leftarrow \text{leere Routen}$ 
while  $U \neq \emptyset$  do
  for each  $r \in U$ :
    for each Fahrzeug  $v$ :  $c_v(r) \leftarrow \min_p c_p(r)$  (beste Position in  $v$ )
    bestimme  $c_r^{(1)}$  (kleinster Fahrzeug-Bestwert) und  $c_r^{(k)}$  ( $k$ -kleinster);  $\text{regret}_k(r) \leftarrow c_r^{(k)} - c_r^{(1)}$ 
    if alle  $c_r^{(1)} = +\infty$  then break
     $r^* \leftarrow \arg \max_{r \in U, c_r^{(1)} < +\infty} \text{regret}_k(r)$  (Ties: kleineres  $c_r^{(1)}$ , dann kleinster Index)
    füge  $r^*$  an bester Position in  $\mathcal{S}$  ein;  $U \leftarrow U \setminus \{r^*\}$ 
return  $\mathcal{S}, U$ 

```

Abb. 4.3: Legende der in den Algorithmen verwendeten Symbole.

R	Menge aller Anfragen
$U \subseteq R$	Menge der noch ungeplanten Anfragen
$\mathcal{S}$	aktuelle Lösung (Menge der Fahrzeugrouten)
r	eine Anfrage
r^*	die im aktuellen Schritt ausgewählte Anfrage
v	ein Fahrzeug
p	eine Einfügeposition (Sublinie und Stelle in der Sublinie)
$c_p(r)$	Einfügekosten für Anfrage r an Position p
$c_v(r)$	beste Position für r in Fahrzeug v : $c_v(r) = \min_p c_p(r)$
$c_r^{(1)} \leq \dots \leq c_r^{(\kappa)}$	sortierte Fahrzeug-Bestwerte $c_v(r)$ für r
$c_r^{(k)}$	Kosten des k -besten Fahrzeugs für r
$\text{regret}_k(r)$	Regret-Wert: $c_r^{(k)} - c_r^{(1)}$
κ	Anzahl der Fahrzeuge
$m = R $	Anzahl der Anfragen

Laufzeitkomplexität. Beide Verfahren sind gierig-iterativ: Sie probieren keine Einplanungsreihenfolgen durch, sondern bewerten in jedem Schritt nur den aktuellen Zustand und planen genau eine Anfrage endgültig ein. Daraus ergibt sich die Laufzeit aus drei Faktoren: höchstens m Iterationen (pro Schritt wird eine Anfrage eingeplant oder das Verfahren endet), je Iteration bis zu m offene Anfragen, und je Anfrage eine mit der Zahl der bereits eingeplanten Halte wachsende Menge an Einfügepositionen (Abschnitt 4.2); zusammen $\mathcal{O}(m^3)$ Kandidatenbewertungen, dieselbe Größenordnung, die Diana und Dessouky [DD04] für ihre Regret-Insertion angeben. Da jede Kandidatenbewertung zusätzlich die Restroute des Fahrzeugs zeitlich fortschreibt (Abschnitt 4.1.3), kommt im ungünstigsten Fall ein weiterer Faktor der Routenlänge hinzu. Auf den Benchmark-Instanzen bleiben die Routen jedoch kurz, sodass dieser Faktor praktisch nicht ins Gewicht fällt. Der Parameter k ändert an alldem nichts: Er beeinflusst nur die Auswahl über die κ sor-

tierten Fahrzeug-Bestwerte, nicht die Zahl der zu bewertenden Positionen: Greedy und Regret- k haben daher dieselbe Komplexität und, wie die Evaluation zeigt, auch praktisch nahezu identische Rechenzeiten.

5 Implementierung

Die Heuristiken sind in Python 3.11 implementiert. Python ist lesbar genug für schnelles Prototyping algorithmischer Ideen, und `numpy` deckt die Matrix-Operationen ab (Distanzmatrix, Kostenberechnungen). Die Laufzeitanforderungen konstruktiver Heuristiken werden auf den betrachteten Instanzgrößen auch von einer interpretierten Sprache erfüllt; eine kompilierte Sprache verspräche geringere absolute Rechenzeiten, ist für die Fragestellung dieser Arbeit jedoch nicht erforderlich.

Der vollständige Quellcode wird zusammen mit der Arbeit abgegeben. Dieses Kapitel beschreibt deshalb keine Code-Details, sondern die Entwurfsentscheidungen: welche Datenstrukturen es gibt, warum sie so geschnitten sind und wie die zentralen Berechnungen ablaufen.

5.1 Kernfragen und Lösungen

Über das reine Einfügen einer Anfrage hinaus prägen einige Kernfragen die Implementierung und die erreichbare Lösungsqualität. Sie werden vorab erläutert, weil sie die Gestaltung der nachfolgenden Datenstrukturen und Operationen bestimmen.

5.1.1 Struktur

Die Route eines Fahrzeugs ist keine einzelne Fahrt, sondern eine *Folge von Sublinien*: maximale Abschnitte, auf denen das Fahrzeug ohne Wende in einer Richtung fährt; aufsteigend oder absteigend. Beim Übergang von einer Sublinie zur nächsten wechselt das Fahrzeug die Richtung (eine Wende). Liegen das Ende der einen Sublinie und der Beginn der nächsten an *verschiedenen* Haltestellen, fährt es dabei zusätzlich leer von der einen zur anderen, eine Leerfahrt (Abbildung 5.1). So endet etwa die aufsteigende Sublinie $1 \rightarrow 3$ bei 3, während die folgende absteigende Sublinie $2 \rightarrow 1$ bei 2 beginnt. Das Fahrzeug wendet bei 3 und fährt leer von 3 nach 2. Treffen Ende und Beginn dagegen zusammen, wendet das Fahrzeug an Ort und Stelle, und es entsteht keine Leerfahrt.

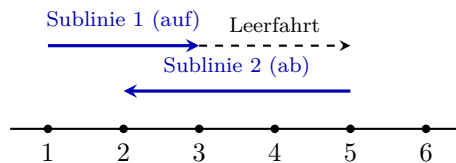


Abb. 5.1: Aufbau einer Fahrzeugroute: eine Folge gerichteter Sublinien (durchgezogen), verbunden durch eine Leerfahrt beim Richtungswechsel (gestrichelt).

Entscheidend ist, dass diese Sublinien *eine durchgehende Zeitachse* bilden: Das Fahrzeug durchläuft sie nacheinander, und wann eine Sublinie beginnt, hängt davon ab, wann es die vorige beendet. Wird eine Anfrage in eine bestehende Sublinie eingefügt, kann diese *länger* werden; und damit verschiebt sich alles, was zeitlich danach kommt.

Ein Beispiel: Ein Fahrzeug fährt zunächst aufsteigend $1 \rightarrow 3$ und danach absteigend $5 \rightarrow 2$. Kommt die Anfrage $2 \rightarrow 4$ hinzu, wächst die erste Sublinie zu $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$, und das Fahrzeug erreicht Haltestelle 5 später als bisher. Die Zeitfenster der neuen Anfrage allein zu prüfen genügt daher nicht; entscheidend ist, ob das Fahrzeug die *gesamte restliche Route* noch fristgerecht schafft.

Lösung. Jede Einfügung wird nicht isoliert geprüft, sondern die komplette Restroute ab der Einfügestelle als eine durchgehende Fahrt nachgerechnet. Die nachfolgenden Sublinien dürfen sich dabei zeitlich nach hinten verschieben; zulässig ist die Einfügung genau dann, wenn dabei kein spätestes Zeitfenster und keine maximale Fahrtzeit reißt. So bleiben auch Einfügungen möglich, bei denen das Fahrzeug etwas später, aber noch rechtzeitig ankommt; wie es auch die exakte MILP-Lösung [RSS24] zulässt bzw. wie es durch die Serviceversprechen erlaubt wird.

5.1.2 Leerfahrten

Zwischen dem Ende einer Sublinie und dem Beginn der nächsten fährt das Fahrzeug *leer*. Diese Leerfahrt gehört keiner einzelnen Anfrage, sondern dem Übergang zwischen zwei Sublinien. Sie verursacht aber reale Fahrzeugkilometer und ist damit ein *Teil der Einfügekosten*: Wie teuer eine Anfrage ist, hängt direkt davon ab, wie weit das Fahrzeug für sie leer fahren muss. Vor der *ersten* Sublinie einer Route gibt es dagegen keine Leerfahrt: In diesem Modell werden Anfahrtkosten vom Depot nicht gerechnet. Die Route beginnt direkt an der ersten Abholhaltestelle (Abschnitt 5.1.3).

Am deutlichsten ist das beim *Eröffnen* einer neuen Sublinie: Um die Anfrage überhaupt zu erreichen, muss das Fahrzeug zunächst leer dorthin fahren. Diese Anfahrt ist reine Leerstrecke (Abbildung 5.2).

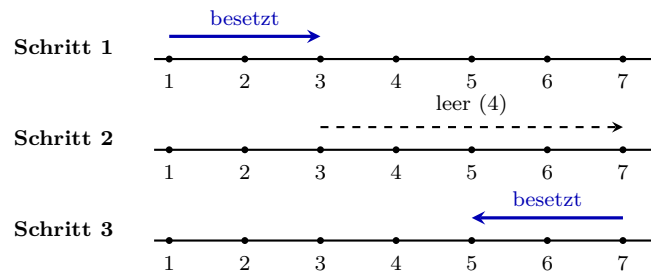


Abb. 5.2: Eröffnen einer neuen Sublinie für die Anfrage $7 \rightarrow 5$. *Schritt 1:* Das Fahrzeug beendet die aufsteigende Sublinie $1 \rightarrow 3$. *Schritt 2:* Es fährt leer von 3 nach 7 (Anfahrt zum Pickup); diese vier Leerkilometer sind der Kostenanteil. *Schritt 3:* Es bedient die neue absteigende Sublinie $7 \rightarrow 5$.

Beispielrechnung. Bei einem Haltestellenabstand von je einer Einheit setzt sich der Zusatzfahraufwand der Anfrage $7 \rightarrow 5$ aus zwei Teilen zusammen:

- *Leerfahrt (Anfahrt):* $3 \rightarrow 7$, also 4 Einheiten;
- *Direktfahrt der Anfrage:* $7 \rightarrow 5$, also 2 Einheiten.

Der Zusatzfahraufwand beträgt $4 + 2 = 6$, davon sind 4 reine Leerstrecke. Genau diese Leerkilometer machen die Anfrage teuer und müssen in die Einfügekosten eingehen, damit die Regret-Heuristik sie richtig bewertet; eine Anfrage, deren Pickup weiter weg liegt, wird so entsprechend teurer.

Liegt eine Anfrage dagegen auf einer Strecke, die das Fahrzeug ohnehin schon befährt, ist sie günstig: Sie fährt auf bereits gezählter Strecke mit und verursacht kaum zusätzliche Leerkilometer.

Lösung. Der Zusatzfahraufwand einer Einfügung umfasst daher nicht nur die Mehrkilometer innerhalb der Sublinie, sondern auch die Änderung der Leerfahrten zu den angrenzenden Sublinien. So bewertet die Regret- k -Heuristik jede Anfrage mit ihren *tatsächlichen* Mehrkilometern. Dieselben Leerfahrten werden auf Ebene der **VehicleRoute**, der Datenstruktur, die die vollständige Route eines Fahrzeugs hält (Abschnitt 5.2.5), in

die gesamte gefahrene Strecke aufgenommen, jeweils vom Ende einer Sublinie zum Beginn der nächsten, sodass Einfügekosten, Zielfunktion und die Kennzahl der gefahrenen Leerkilometer konsistent dieselben Fahrzeugkilometer zugrunde legen.

5.1.3 Depot

Wo startet ein Fahrzeug, bevor es seine erste Anfrage bedient? Gibt es ein Depot, von dem aus es anfährt?

Im zugrunde liegenden Modell [RSS24] gibt es zwar einen *Depot-Knoten* (Knoten 0), der den Fahrzeugrouten als Quelle und Senke dient; seine *Distanz und Fahrzeit zu jeder Haltestelle sind jedoch 0*. Das Depot besitzt also keinen Ort im Liniennetz und verursacht keine Anfahrt; effektiv dürfen die Fahrzeuge ihre Route an einer beliebigen Haltestelle kostenlos beginnen und beenden. Das Modell der ereignisbasierten Formulierung erzwingt diese Eigenschaft sogar explizit, indem Distanz und Fahrzeit zum Depot null sein müssen. Auch die Benchmark-Instanzen enthalten keine Depotposition, sondern nur Haltestellen und Anfragen.

Entscheidung. Da das Depot keine Anfahrtskosten verursacht, wird es in dieser Arbeit weggelassen: Die erste Sublinie eines Fahrzeugs beginnt unmittelbar an der Abholhaltestelle ihrer ersten Anfrage. So bewertet die Heuristik dieselben Fahrzeugkilometer wie der MILP-Solver, und der Vergleich der Optimalitätslücke bleibt fair.

Das eigentliche Startproblem liegt woanders: Am Anfang sind alle Fahrzeuge leer und damit *gleich*, sodass die Heuristik keinen Anhaltspunkt hat, welches Fahrzeug womit beginnen soll. Ein Depot würde daran nichts ändern, da es alle Fahrzeuge gleich behandelt. Dieses Initialisierungsproblem wird stattdessen über das *Seeding* der Startanfragen gelöst, das im folgenden Abschnitt behandelt wird.

5.1.4 Initialisierung

Solange alle Fahrzeuge leer sind, kostet das Eröffnen der ersten Sublinie in jedem Fahrzeug gleich viel. Es gibt ja kein Depot, das die Fahrzeuge unterscheidet. Für jede Anfrage gilt deshalb $c_r^{(1)} = c_r^{(k)}$ und damit $\text{regret}_k(r) = 0$. Der Regret-Wert kann am Anfang also nichts unterscheiden; ohne Eingriff füllt der Algorithmus die Fahrzeuge in einer zwar deterministischen, aber inhaltlich unbegründeten Reihenfolge (Tie-Break nach Anfrageindex, Abschnitt 4.2) und neigt dazu, ein Fahrzeug zu beladen, bevor die übrigen starten.

Um das Verfahren in Gang zu bringen, erhält *jedes* Fahrzeug vor der Einfügeschleife aus Abschnitt 4.4 eine *Startanfrage* (einen *Seed*), eine bewusste Entscheidung: Es wird nicht nur ein einzelnes Fahrzeug angestoßen, sondern alle κ gleichzeitig. Erst dadurch unterscheiden sich die Fahrzeuge, der Regret-Wert wird aussagekräftig, und die anfängliche räumliche wie zeitliche Verteilung der Routen wird festgelegt. Das Seeding gehört damit zur Regret- k -Insertion. Welche Anfragen als Seeds dienen, lässt sich auf verschiedene Weisen bestimmen:

- **Zeitbasiert:** die κ Anfragen mit den frühesten Abfahrtszeitpunkten. Einfach und naheliegend; die Fahrzeuge starten mit den zeitlich dringendsten Anfragen. Liegen diese räumlich nah beieinander, droht allerdings eine Häufung der Fahrzeuge in einem Linienabschnitt.
- **Räumlich isoliert:** die räumlich am stärksten isolierten Anfragen zuerst (Dezentralisierungsindex nach Diana und Dessouky [DD04]). Schwer einzufügende, abgelegene Anfragen werden so früh gesichert, solange noch Platz ist.
- **Längste Anfragen:** die Anfragen mit der größten Direktstrecke zuerst. Jedes Fahrzeug startet damit mit einer großen Sublinie, in die sich viele kürzere, überlappende Anfragen später nahezu ohne Mehrweg einfügen lassen; lange Anfragen tragen zudem am meisten zur eingesparten Distanz bei und sind später schwerer unterzubringen.
- **Richtungsbalanciert:** ein Teil der Fahrzeuge mit aufsteigenden, ein Teil mit absteigenden Startanfragen, passend zur Direktionalität des liDARP.

Unabhängig vom Auswahlkriterium wird je Fahrzeug *genau ein* Seed vergeben, sodass alle κ Fahrzeuge gleichzeitig starten und nicht zunächst alle Anfragen in dasselbe Fahrzeug wandern. Diese Entscheidung hat einen möglichen Nachteil: Die κ ausgewählten Anfragen werden zwangsweise auf verschiedene Fahrzeuge verteilt. Gehörten zwei von ihnen eigentlich in dieselbe Route (etwa die zwei längsten Anfragen, wenn sie sich gut überlappen), ist diese ungünstige Trennung bereits vor der ersten Iteration festgeschrieben und wird später nicht mehr revidiert. Die Diskussion greift diesen Punkt auf.

Konkret laufen alle vier Kriterien auf dasselbe Muster hinaus: Die Anfragen werden nach der jeweiligen Kennzahl *sortiert*, und die vordersten werden (eine je Fahrzeug) als Seeds genommen. Weil es kein Depot gibt, sind vor der Zuweisung alle Fahrzeuge gleich; welcher Seed in welches Fahrzeug kommt, ist daher beliebig (der erste Seed ins

erste Fahrzeug, der zweite ins zweite und so fort). Jeder Seed eröffnet die erste Sublinie seines Fahrzeugs; weil das Fahrzeug leer ist, kostet das Eröffnen nichts. Gibt es weniger Anfragen als Fahrzeuge, bleiben die überzähligen Fahrzeuge leer und werden in der Schleife regulär gefüllt.

Entscheidung. Welches dieser Kriterien am besten geeignet ist, lässt sich nicht von vornherein bestimmen. Die Wahl der Startanfragen kann das Endergebnis erheblich beeinflussen. Daher legt diese Arbeit sich nicht auf ein einzelnes Verfahren fest, sondern *prüft alle vier* in den Experimenten in Abschnitt 6.2 gegeneinander und untersucht ihren Einfluss auf die Lösungsqualität (den Zielfunktionswert).

5.1.5 Einfügepositionen und beste Einfügeposition

Bevor eine Anfrage eingeplant wird, ist zu bestimmen, an welchen Stellen sie überhaupt eingefügt werden kann: die Menge ihrer *Einfügepositionen*. Sie ergibt sich aus vier Bedingungen.

1. Richtung. Eine Anfrage kann nur in eine Sublinie ihrer *eigenen* Richtung eingefügt werden: eine aufsteigende Anfrage nur in aufsteigende Sublinien, eine absteigende nur in absteigende (Direktionalität). Sublinien der Gegenrichtung scheiden aus. Zusätzlich lässt sich stets eine *neue* Sublinie eröffnen, als einzelne Sublinie an jeder Stelle der Route, zu der das Fahrzeug leer übersetzt. Die Kandidatenmenge besteht also aus allen gleichgerichteten Sublinien plus der Option, eine neue anzulegen (Abbildung 5.3).

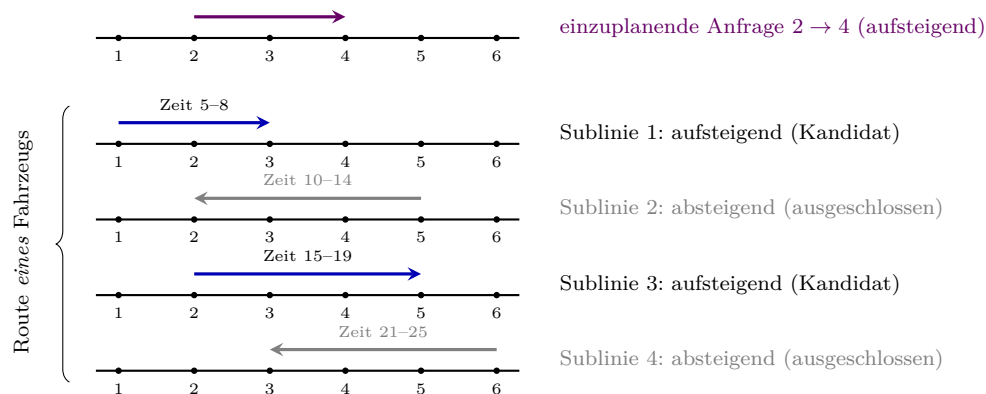


Abb. 5.3: Einfügepositionen für die aufsteigende Anfrage $2 \rightarrow 4$ (oben, violett). Darunter die Route eines Fahrzeugs: vier Sublinien, die das Fahrzeug zeitlich nacheinander abfährt (Bedienintervalle jeweils angegeben; zwischen den Sublinien wendet es leer). Kandidaten sind die aufsteigenden Sublinien 1 und 3 sowie das Eröffnen einer neuen Sublinie; die absteigenden Sublinien 2 und 4 scheiden wegen der Richtung aus. Unter den Kandidaten entscheidet anschließend die Zeitprüfung, welche Position tatsächlich zulässig ist.

2. Kapazität. Eine Anfrage befördert q_r Fahrgäste (es dürfen mehrere sein) und belegt damit auf ihrem gesamten Fahrabschnitt q_r Plätze. Eine Position ist nur zulässig, wenn die Fahrzeuglast (die Summe aller gleichzeitig an Bord befindlichen Fahrgäste) an keinem Halt die Kapazität überschreitet. Auf einem bereits ausgelasteten Abschnitt scheidet sie aus, auch wenn sie zeitlich passt. Treffen Aus- und Einstieg an derselben Haltestelle zusammen, gilt dabei die Reihenfolge aus Abschnitt 2.4: erst aussteigen, dann einsteigen. Die Lastprüfung folgt dieser Reihenfolge, indem sie die Last nach jeder einzelnen Aktion prüft. Da sich die Last zwischen zwei Halten nicht ändert, deckt die Prüfung an den Halten zugleich jeden Streckenabschnitt dazwischen ab. Dieser Prüfung ist eine *Vorabprüfung* vorgeschaltet: Übersteigt schon der Bedarf q_r einer einzelnen Anfrage die Kapazität, gibt es in *keinem* Fahrzeug und an *keiner* Position einen gültigen Platz; eine solche Anfrage ist von vornherein unzulässig und wird ohne Einfügeversuch abgelehnt.

3. Eigene Zeitfenster. Unter den gleichgerichteten Sublinien entscheidet meist die Zeit, ob eine Position gültig ist: Pickup und Dropoff müssen sich in die Zeitfolge der Sublinie einfügen lassen, ohne die Zeitfenster der Anfrage selbst (frühester Abfahrts-, spätester Ankunftszeitpunkt, maximale Fahrtzeit) und das β -Serviceversprechen (Abholung kurz nach dem frühesten Abfahrtszeitpunkt) zu verletzen.

4. Verträglichkeit mit dem Rest der Route. Schließlich darf die Einfügung *alles danach* nicht unzulässig machen: Wie in Abschnitt 5.1.1 beschrieben, kann sie die Sublinie verlängern und nachfolgende Sublinien zeitlich nach hinten verschieben. Eine Position ist nur dann gültig, wenn die gesamte restliche Route weiterhin alle Zeitfenster einhält.

Nur Positionen, die alle vier Bedingungen erfüllen, kommen in Frage. Da diese Bedingungen die Zahl der Positionen auf eine überschaubare Menge begrenzen, lässt sich für jede Anfrage (und zwar je Fahrzeug) stets die *beste* (günstigste) Einfügeposition bestimmen. Dieser Fahrzeug-Bestwert ist die Grundlage für die Auswahl per Regret- k (Vergleich der besten Fahrzeuge).

Gleichstand. Die günstigste Position ist allerdings nicht immer eindeutig. *Beispiel:* Passt eine Anfrage in zwei Sublinien, ohne eine davon zu verlängern, verursacht sie in beiden keinen Mehrweg (Kosten 0). Beide Positionen sind gleich gut. In solchen Fällen entscheidet ein *Tie-Breaker*, der deterministisch und damit reproduzierbar gewählt wird:

1. Es wird die Position mit dem *früheren Pickup-Zeitpunkt* bevorzugt; der Fahrgast wartet weniger, ohne dass sich Kosten oder Zielfunktionswert ändern.
2. Bleibt es danach gleich, entscheidet die kleinste Sublinien-Nummer (ein neu zu eröffnender Platz wird zuletzt gereiht).

So liefert die Suche für jedes Fahrzeug eine eindeutige beste Einfügeposition.

5.1.6 Verlängern einer Sublinie und die Wirkung auf die Nachbarn

Im günstigsten Fall liegen Start und Ziel einer Anfrage bereits auf dem Weg einer passenden Sublinie: Das Fahrzeug fährt die Strecke ohnehin, die Anfrage steigt nur zu, und es entsteht *kein* Mehrweg ($\Delta = 0$). Oft aber reicht die Sublinie nicht ganz bis zur Anfrage. Dann muss sie *verlängert* werden, entweder *vorne* (der Pickup liegt vor ihrem bisherigen Start) oder *hinten* (der Dropoff liegt hinter ihrem bisherigen Ende). Eine solche Verlängerung betrifft je Ende genau einen Nachbarn in der Routenfolge des Fahrzeugs (vgl. Abschnitt 5.1.1 und 5.1.2):

- Verlängerung *vorne* (neuer Startpunkt): Es ändert sich der Übergang von der *vorhergehenden* Sublinie, also die Leerfahrt, mit der das Fahrzeug zur verlängerten Sublinie anreist.
- Verlängerung *hinten* (neues Ende): Es ändert sich der Übergang zur *nachfolgenden* Sublinie; die Leerfahrt, mit der das Fahrzeug danach weiterfährt.

Wie teuer das ist, hängt davon ab, wo die jeweils betroffene Nachbar-Sublinie wendet. Das zerfällt je Richtung in drei Unterfälle, die im Folgenden an einem Beispiel illustriert

werden. Die beiden Richtungen schließen sich dabei nicht aus: Verlängert eine Anfrage die Sublinie an *beiden* Enden, treten beide Wirkungen gemeinsam auf (Fall 3).

Fall 1: Verlängerung am Anfang. *Beispiel:* Die aufsteigende Sublinie $3 \rightarrow 5$ nimmt die Anfrage $2 \rightarrow 4$ auf und wird zu $2 \rightarrow 5$. Ihr Start rutscht von 3 auf 2. Betroffen ist die *vorhergehende* (absteigende) Sublinie, die bei einer Station T wendet; *vor* und *hinter* beziehen sich im Folgenden auf deren Fahrtrichtung (Abbildung 5.4):

- (1a) **T liegt vor dem neuen Start** (die vorige Sublinie endet bei 3 und wendet bisher direkt in $3 \rightarrow 5$). Das Fahrzeug steht am Wendepunkt bei 3, muss nun aber leer nach 2 und besetzt wieder zurück: Aus $\dots \rightarrow 4 \rightarrow 3$, Wende, $3 \rightarrow 4 \rightarrow 5$ wird $\dots \rightarrow 4 \rightarrow 3 \rightarrow 2$ (leer), Wende, $2 \rightarrow 3 \rightarrow 4 \rightarrow 5$. Es entsteht eine *neue* Leerfahrt $3 \rightarrow 2$, insgesamt ein Umweg von 2 Einheiten (leer hin, besetzt zurück). *Echter Mehrweg*.
- (1b) **T trifft genau den neuen Start** (die vorige Sublinie endet bereits bei 2). Bisher gab es eine Leerfahrt $2 \rightarrow 3$ zum alten Start; jetzt wendet das Fahrzeug direkt bei 2, und diese Leerfahrt *entfällt*. Das früher leer gefahrene Stück $2 \rightarrow 3$ wird nun besetzt zurückgelegt; kein zusätzlicher Kilometer ($\Delta = 0$).
- (1c) **T liegt hinter dem neuen Start** (die vorige Sublinie reicht bis 1). Das Fahrzeug fuhr ohnehin durch 2 hindurch; die bestehende Leerfahrt $1 \rightarrow 3$ *schrumpft* auf $1 \rightarrow 2$, das Stück $2 \rightarrow 3$ wird von leer auf besetzt umgebucht; ebenfalls kein zusätzlicher Kilometer ($\Delta = 0$).

Die Verlängerung am Anfang ist also nur dann teuer, wenn die vorige Sublinie schon vor dem neuen Pickup endet (Fall 1a); erreicht sie ihn oder fährt sie darüber hinaus (Fälle 1b und 1c), fährt das Fahrzeug die Strecke ohnehin und die Verlängerung ist gratis.

— verlängerte Sublinie — Nachbar-Sublinie - - Leerfahrt

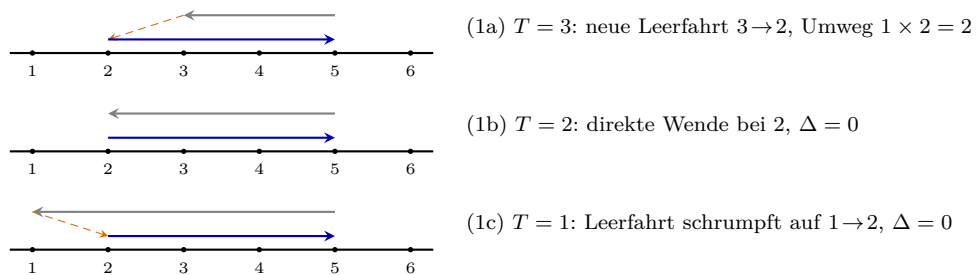


Abb. 5.4: Verlängerung einer aufsteigenden Sublinie am Anfang (Pickup 2, neue Sublinie $2 \rightarrow 5$, blau). Je nachdem, wo die vorhergehende absteigende Sublinie (grau) wendet, entsteht eine *neue* Leerfahrt (1a, orange), eine *direkte* Wende ganz ohne Leerfahrt (1b) oder eine *verkürzte* Leerfahrt (1c). Nur 1a verursacht echten Mehrweg; die Lücke wird dabei *doppelt* durchfahren (leer hinunter zum Pickup, besetzt wieder hinauf), daher $1 \times 2 = 2$.

Fall 2: Verlängerung am Ende. Die Verlängerung am Ende (der Dropoff liegt hinter dem bisherigen Sublinienende) verhält sich spiegelbildlich: Betroffen ist die *nachfolgende* Sublinie, und dieselben drei Unterfälle treten bezogen auf ihren Startpunkt U auf. Echter Mehrweg entsteht nur, wenn die nachfolgende Sublinie vor dem neuen Sublinienende startet (auf dem bereits durchfahrenen Abschnitt). Dann wird die Verlängerung doppelt durchfahren (besetzt hin, leer zurück zum Anschluss); startet sie am neuen Ende oder dahinter, entfällt oder schrumpft die bestehende Leerfahrt und die Verlängerung ist gratis ($\Delta = 0$).

Fall 3: Verlängerung an beiden Enden. Liegt der Pickup vor dem bisherigen Start *und* der Dropoff hinter dem bisherigen Ende (etwa die Anfrage $2 \rightarrow 7$ für die Sublinie $3 \rightarrow 5$), treten Fall 1 und Fall 2 gemeinsam auf: Am neuen Start ist die vorhergehende, am neuen Ende die nachfolgende Sublinie betroffen, und an jedem Ende gelten die jeweiligen Unterfälle unabhängig voneinander. Die Konsequenzen addieren sich: Der Zusatzfahraufwand ist die Summe der Effekte an beiden Übergängen. Im günstigsten Fall sind beide gratis ($\Delta = 0$), im ungünstigsten entstehen an beiden Enden neue Leerfahrten. Für die Kostenrechnung ist das unproblematisch, da die Streckendifferenz aus Abschnitt 5.3.2 alle Kombinationen ohne Fallunterscheidung erfasst.

Alle drei Fälle zeigen dasselbe Muster: Die reinen Mehrkilometer *innerhalb* der Sublinie erfassen nur einen Teil der Kosten; entscheidend ist, ob die Verlängerung beim betroffenen Nachbarn eine *neue* Leerfahrt erzwingt (teuer), eine bestehende *auföst* (gratis) oder nur *verkürzt* (gratis). Genau diese Änderung der Leerfahrten steckt im Zusatzfahraufwand der Einfügekosten (Abschnitt 5.1.2); weshalb dieselbe Anfrage je nach Nachbar-Sublinie unterschiedlich viel kostet.

5.1.7 Annahme der Anfrage

Ob eine Anfrage überhaupt bedient wird, ergibt sich aus der *Zielfunktion*. Sie maximiert eine gewichtete Summe aus zwei Größen:

$$w_1 \cdot (\text{Anzahl bedienter Fahrgäste}) + w_2 \cdot (\text{eingesparte Distanz}),$$

wobei die eingesparte Distanz die Summe der Direktstrecken der bedienten Anfragen abzüglich der tatsächlich gefahrenen Fahrzeugkilometer ist. Die konkreten Gewichte werden im Versuchsaufbau der Experimente festgelegt (Abschnitt 6.1).

Wann sich eine Anfrage lohnt. Eine Anfrage an Position p zu bedienen, ändert die Zielfunktion um

$$w_1 \cdot q_r + w_2 \cdot (t_{o_r, d_r} - \Delta_p(r)),$$

mit dem Zusatzfahraufwand $\Delta_p(r)$ aus Abschnitt 4.3.3: Sie bringt $w_1 \cdot q_r$ für ihre q_r Fahrgäste, kostet aber jeden über ihren Direktweg t_{o_r, d_r} hinaus gefahrenen Kilometer (Umweg oder Leerfahrt) mit w_2 . Die Bedienung erhöht den Zielfunktionswert also genau dann, wenn der Mehraufwand über den Direktweg hinaus, $\Delta_p(r) - t_{o_r, d_r}$, unter der Schwelle $w_1 \cdot q_r / w_2$ bleibt; darüber senkt sie den Wert.

Entscheidung. In jedem Schritt kommen nur Anfragen in Frage, deren beste Einfügung den Zielfunktionswert *erhöht* (negative Einfügekosten); unter ihnen wählt die Heuristik die Anfrage mit dem größten Regret-Wert. Eine Anfrage, deren günstigste Einfügung den Wert aktuell senken würde, wird in diesem Schritt nicht angenommen.

Verhältnis zur allgemeinen Auswahlregel. Die Pseudocodes in Abschnitt 4.4 wählen in jedem Schritt unter allen *zulässigen* Anfragen (bestes Fahrzeug mit endlichen Kosten). Die hier getroffene Entscheidung verschärft das: Zulässigkeit allein genügt nicht, die beste Einfügung muss den Zielwert *erhöhen*. Beide Regeln fallen meist zusammen, denn eine zulässige Einfügung senkt den Zielwert erst, wenn ihr Mehrweg die Schwelle $w_1 \cdot q_r / w_2$ übersteigt. Sie unterscheiden sich also nur im Randfall einer zulässigen, aber wertsenkenden Einfügung, die diese Arbeit nicht einplant, sondern aufschiebt.

Eine solche Ablehnung ist jedoch *nicht endgültig*: Da sich die Routen mit jeder Einfügung ändern, wird die Anfrage in den folgenden Schritten erneut bewertet. Hat sich durch zwischenzeitliche Einfügungen eine Sublinie verlängert, kann ihr Umweg unter die Schwelle fallen, sodass sie in einer späteren Iteration doch noch angenommen wird. Erst wenn keine Anfrage mehr eine wertsteigernde Einfügung besitzt, endet das Verfahren, und die verbliebenen Anfragen werden endgültig abgelehnt.

So verfolgt die Heuristik dieselbe Zielfunktion wie die exakte Lösung und nimmt nicht jede zulässige Anfrage an. Das macht ihre Akzeptanzrate mit der der MILP-Lösung vergleichbar.

Die Gewichte w_1 und w_2 sind damit der Hebel zwischen Akzeptanz und gefahrener Distanz: Ein höheres w_1 (oder kleineres w_2) verschiebt die Schwelle nach oben, sodass sich mehr Anfragen lohnen; ein kleineres w_1 (oder höheres w_2) gewichtet die eingesparte Distanz stärker und lehnt teure Anfragen früher ab.

5.1.8 Regret- k -Kosten

Der Parameter k liegt zwischen 2 (Vergleich mit dem zweitbesten Fahrzeug) und der Fahrzeuganzahl κ (Vergleich mit dem schlechtesten). Unabhängig von k werden in jedem Schritt für *jedes* Fahrzeug die Einfügekosten einer Anfrage bestimmt. Das ist auch unvermeidlich: Ohne die Kosten aller Fahrzeuge ließe sich gar nicht sagen, welches das zweit- oder allgemein das k -beste Fahrzeug ist. Nur der anschließende Vergleich ändert sich.

Aufbau der Berechnung. Die Regret-Kosten einer Anfrage r entstehen in zwei Stufen:

1. **Fahrzeug-Bestwerte:** Für jedes der κ Fahrzeuge v werden alle Einfügepositionen von r (Abschnitt 5.1.5) geprüft; die günstigste liefert den *Fahrzeug-Bestwert* $c_v(r)$ (Notation aus Abschnitt 4.4).
2. **Sortierung:** Die κ Fahrzeug-Bestwerte werden aufsteigend sortiert; der i -t kleinste heißt $c_r^{(i)}$, also $c_r^{(1)} \leq c_r^{(2)} \leq \dots \leq c_r^{(\kappa)}$.

Der *Regret-Wert* ist die Differenz zwischen dem k -besten und dem besten Fahrzeug:

$$\text{regret}_k(r) = c_r^{(k)} - c_r^{(1)}.$$

Da der Beitrag $w_1 q_r$ und die Direktstrecke einer Anfrage über alle Fahrzeuge gleich sind, unterscheiden sich die Fahrzeug-Bestwerte nur im Zusatzfahraufwand. Für den Vergleich genügt daher dieser.

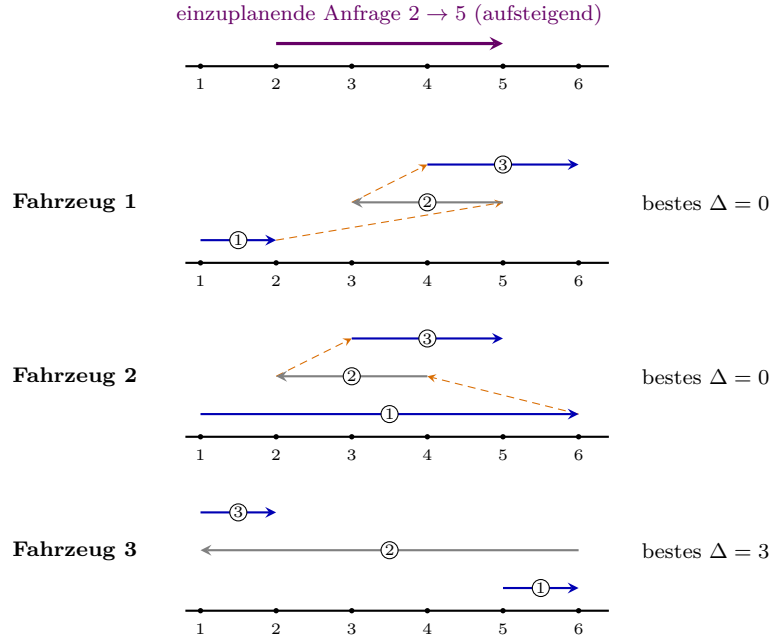


Abb. 5.5: Oben die einzuplanende aufsteigende Anfrage $2 \rightarrow 5$ (violett), darunter die Fahrzeuge 1 bis 3 (in der Notation aus Abschnitt 4.4: v_1 bis v_3) mit je drei Sublinien (aufsteigend blau, absteigend grau) auf sechs Haltestellen. Die Pfeilnummern 1, 2, 3 geben die zeitliche Reihenfolge der Sublinien an; die gestrichelten Pfeile sind die Leerfahrten beim Richtungswechsel (vom Ende einer Sublinie zum Beginn der nächsten). Die Positionssuche liefert für die Anfrage je Fahrzeug einen besten Zusatzfahraufwand; aus diesen drei Werten ergeben sich Regret-Wert und Fahrzeugwahl.

Beispiel. Die aufsteigende Anfrage $2 \rightarrow 5$ soll eingeplant werden. Es gibt drei Fahrzeuge mit je drei Sublinien (Abbildung 5.5). Bei Fahrzeug v_1 verläuft bereits eine Leerfahrt von 2 nach 5. Die Anfrage füllt sie nur auf, ohne dass ein einziger Mehrkilometer entsteht. Bei v_2 deckt die besetzte aufsteigende Sublinie $1 \rightarrow 6$ die Haltestellen 2 und 5 ohnehin ab; auch hier entsteht kein Umweg. Bei v_3 dagegen liegen die beiden aufsteigenden Sublinien an den Rändern: $5 \rightarrow 6$ erreicht nur die Zielhaltestelle 5, $1 \rightarrow 2$ nur die Starthaltestelle 2, und keine verbindet beide. Um die Anfrage zu bedienen, muss eine Sublinie verlängert werden; führt man etwa $1 \rightarrow 2$ besetzt bis 5 weiter, kommen 3 zusätzliche *besetzte* Kilometer (die Strecke von 2 bis 5) hinzu. Die Positionssuche liefert also

$$c_{v_1}(r) = 0, \quad c_{v_2}(r) = 0, \quad c_{v_3}(r) = 3.$$

Aufsteigend sortiert ergibt das $c_r^{(1)} = 0$, $c_r^{(2)} = 0$, $c_r^{(3)} = 3$ und damit

$$\text{regret}_2 = c_r^{(2)} - c_r^{(1)} = 0 - 0 = 0, \quad \text{regret}_3 = c_r^{(3)} - c_r^{(1)} = 3 - 0 = 3.$$

Wird die Anfrage ausgewählt, kommt sie in eines ihrer beiden kostenlosen Fahrzeuge. Bei diesem Gleichstand entscheidet der Tie-Breaker aus Abschnitt 5.1.5 (hier v_1).

Warum sich ein größeres k lohnt. Das Beispiel illustriert am konkreten Routenbild den in Abschnitt 4.3.1 beschriebenen Effekt: Bei $k = 2$ wirkt die Anfrage unkritisch (zwei kostenlose Fahrzeuge, Regret 0), erst $k = 3$ deckt auf, dass nach den beiden kostenlosen Optionen nur noch eine teure Alternative bleibt. Passt eine Anfrage nur in weniger als k Fahrzeuge, ist $c_r^{(k)} = +\infty$ und sie erhält maximalen Regret (Abschnitt 4.3.1). Sie wird eingeplant, solange ihre wenigen tauglichen Fahrzeuge noch frei sind.

5.2 Datenstrukturen

Die Implementierung trennt die *Anfrage* (der ursprüngliche Fahrtwunsch eines Fahrgastes) von der *Bedienung* (die konkrete Pickup- oder Dropoff-Aktion in einer Sublinie). Eine **Request** beschreibt eine Anfrage und ist unveränderlich; **Stop**, **Subline**, **VehicleRoute** und **Solution** bilden die Lösungsstruktur. Die **Instance** klammert das gesamte Eingabeproblem zusammen: alle Anfragen, die Distanzmatrix und die Parameter wie Kapazität, α und β .

Die Datenstrukturen-Hierarchie im Überblick:

- **Request**: eine Anfrage (Origin, Destination, Zeitfenster, Fahrgastzahl q_r)
- **Instance**: das gesamte Eingabeproblem (alle Requests, Distanzmatrix, Parameter)
- **Stop**: eine atomare Pickup- oder Dropoff-Aktion einer Anfrage an einer Haltestelle
- **Subline**: eine geordnete Folge von Stops in einer Richtung
- **VehicleRoute**: alle Sublinien eines Fahrzeugs
- **Solution**: alle **VehicleRoute**-Objekte zusammen

Jede Struktur wird im Folgenden mit ihren Feldern beschrieben; Abschnitt 5.2.8 fasst die Hierarchie mit ihren Beziehungen am Ende zusammen. Der vollständige Quellcode liegt der Arbeit bei.

5.2.1 Request

Eine **Request** repräsentiert eine Anfrage: den Fahrtwunsch von q_r gemeinsam reisenden Fahrgästen von einer Start- zu einer Zielhaltestelle in einem definierten Zeitfenster. Sie ist im Lösungsverfahren unveränderlich. Die Datenstruktur beschreibt nur den Wunsch, nicht die konkrete Bedienung.

Eine Anfrage trägt die folgenden Felder:

Feld	Bedeutung
<code>idx</code>	laufender Index der Anfrage
<code>origin, destination</code>	Start- und Zielhaltestelle (o_r, d_r)
<code>demand</code>	Anzahl der Fahrgäste q_r
<code>service_time</code>	Bedienzeit am Halt (Ein-/Aussteigen)
<code>earliest_pickup, latest_pickup</code>	Zeitfenster der Abholung
<code>earliest_dropoff, latest_dropoff</code>	Zeitfenster der Ablieferung
<code>max_ride_time</code>	maximale Fahrtzeit $\alpha \cdot t_{o_r, d_r}$

Alle Felder werden beim Einlesen der Instanzdatei gesetzt und bleiben danach unverändert.

Die Felder `origin` und `destination` bestimmen die Richtung der Anfrage. Bei `origin < destination` ist sie aufsteigend (`is_ascending` liefert `True`), sonst absteigend. Diese Richtung legt fest, in welche Sublinien die Anfrage später eingefügt werden darf.

Bedienzeit vs. Serviceversprechen. Zwei Zeitgrößen dieser Tabelle sind leicht zu verwechseln, gehen aber an unterschiedlichen Stellen in die Zulässigkeitsprüfung ein. `service_time` bezeichnet die Bedienzeit am Halt, also wie lange das Fahrzeug an einer Haltestelle steht, damit Fahrgäste ein- oder aussteigen können. Sie fällt *fix pro Halt* an, unabhängig von der Fahrgastzahl q_r , und entspricht der Bedienzeit b_r aus Abschnitt 2.4. Die Serviceversprechen α und β aus Abschnitt 2.4 sind dagegen globale Qualitätsgarantien für alle Anfragen (maximaler Fahrtzeitfaktor und maximale Abweichung vom Wunschzeitpunkt), keine Aufwandsgrößen.

Zeitfenster: entweder oder. In der Eingabedatei gibt jede Anfrage genau *einen* Wunschzeitpunkt vor: entweder den frühesten Abholzeitpunkt oder den spätesten Ablieferzeitpunkt. Die jeweils anderen Zeitwerte (`latest_pickup`, `earliest_dropoff` usw.) werden beim Einlesen aus dem gegebenen Wert sowie aus den Serviceversprechen α und β abgeleitet. Das spiegelt die Annahme aus Abschnitt 2.4 wider, dass in der Eingabe nur einer der beiden Wunschzeitpunkte vorgegeben ist. Wichtig ist diese Ableitung für alles Weitere: Erst dadurch stehen alle vier Zeitschranken zur Verfügung, gegen die die Zulässigkeitsprüfung Einfügepositionen später prüft (Abschnitt 5.1.5).

5.2.2 Instance

Eine **Instance** fasst das gesamte Eingabeprobblem in einem Objekt zusammen: alle Anfragen, die Distanzmatrix und die globalen Parameter (Fahrzeugzahl, Kapazität, maximale Routendauer sowie die Serviceversprechen α und β).

Feld	Bedeutung
<code>num_vehicles</code>	Anzahl der Fahrzeuge
<code>num_requests</code>	Anzahl der Anfragen
<code>num_stations</code>	Anzahl der Haltestellen
<code>max_time</code>	maximale Routendauer
<code>capacity</code>	Fahrzeugkapazität $Q_{\max}$
<code>alpha</code>	Serviceversprechen: Fahrtzeitfaktor α (vgl. Abschnitt 2.4)
<code>beta</code>	Serviceversprechen: Zeitfensterabweichung β (vgl. Abschnitt 2.4)
<code>requests</code>	Liste aller Request -Objekte
<code>distances</code>	Distanzmatrix (Fahrzeit zwischen je zwei Haltestellen)

Die **Instance** wird einmal beim Einlesen der Benchmark-Datei erzeugt und danach nicht mehr verändert.

Aufbau der Distanzmatrix. `distances[i][j]` liefert die Fahrzeit von Haltestelle i nach Haltestelle j . Die Matrix ist $(n+1) \times (n+1)$ groß und *1-basiert* indiziert: Zeile und Spalte 0 bleiben ungenutzt, sodass die Haltestellennummern direkt als Index dienen (Haltestelle 1 steht in Zeile 1).

5.2.3 Stop

Ein **Stop** repräsentiert genau *eine* Bedienaktion: einen Pickup *oder* einen Dropoff genau einer Anfrage an einer Haltestelle. Der Name bezeichnet also weniger den Halt des Fahrzeugs als die einzelne Aktion. Sie ist die atomare Einheit der Lösungsrepräsentation.

Feld	Bedeutung
<code>station</code>	Haltestelle (1-basiert)
<code>request_idx</code>	Index (<code>idx</code>) der Anfrage, zu der diese Aktion gehört
<code>is_pickup</code>	Abholung oder Ablieferung
<code>arrival</code>	geplante Ankunftszeit
<code>departure</code>	geplante Abfahrtszeit ($\geq \text{arrival} + \text{service_time}$)
<code>load_after</code>	Fahrzeuglast nach Ausführung genau dieser Aktion

Jede bediente Anfrage erzeugt genau zwei **Stop**-Objekte in der Sublinie: eines für den Pickup, eines für den Dropoff. Der Verweis über `request_idx` ist damit eindeutig: Jeder **Stop** gehört zu genau einer Anfrage. **Stop**-Objekte *ohne* Anfrage gibt es nicht; reine Zwischenhalte, Wenden und Leerfahrten werden nicht als Stops gespeichert, sondern implizit aus der Haltefolge berechnet (Abschnitt 5.2.5). Das Feld `load_after` bezeichnet die Last unmittelbar nach dieser einen Aktion (nicht nach allen Aktionen an der Station). Damit lässt sich an jeder Position in konstanter Zeit ($\mathcal{O}(1)$) die Kapazität prüfen, ohne die vorherigen Aktionen aufaddieren zu müssen.

Stop auf Stop an derselben Haltestelle. Treffen an derselben Haltestelle mehrere Aktionen aufeinander (etwa wenn ein Fahrgast aussteigt und ein anderer einsteigt) werden diese als *getrennte* **Stop**-Objekte sequentiell in der Sublinie geführt: Zwei aufeinanderfolgende Stops an derselben Station sind also ausdrücklich zulässig und bei geteilten Haltestellen der Normalfall. Damit spiegelt die Datenstruktur direkt die Aufeinanderfolge-Regel aus Abschnitt 2.4 wider: an einer geteilten Haltestelle steigt der erste Fahrgast erst vollständig aus, bevor der zweite einsteigt. Ein einziger **Stop** ist also nie gleichzeitig Pickup und Dropoff, auch wenn beide an derselben physischen Haltestelle stattfinden.

5.2.4 Subline

Eine **Subline** ist ein maximal langer, richtungskonsistenter Fahrabschnitt und entspricht direkt dem gleichnamigen Begriff aus Abschnitt 2.4.

Feld	Bedeutung
direction	Richtung: aufsteigend oder absteigend
stops	geordnete Haltefolge (Liste von Stop -Objekten)

Die Richtung einer Sublinie ('**asc**' oder '**desc**') ist unveränderlich. Sie bestimmt gemäß der Direktionalitätseigenschaft (Abschnitt 2.4), welche Anfragen in diese Sublinie eingefügt werden dürfen. Leere Sublinien (noch ohne Halte) können beim Anlegen einer neuen Sublinie erzeugt und anschließend befüllt werden.

Invariante: jede bediente Anfrage erscheint genau zweimal. Wenn eine Anfrage in einer Sublinie bedient wird, taucht ihr **request_idx** in der Stops-Liste genau zweimal auf: einmal mit **is_pickup = True** (Pickup) und einmal mit **is_pickup = False** (Dropoff), wobei der Pickup vor dem Dropoff steht. Diese Invariante stellt die Einfügeoperation sicher, die Pickup und Dropoff stets zusammen in dieselbe Sublinie einfügt (Abschnitt 5.1.5) und eingeplante Anfragen nie wieder entfernt.

5.2.5 VehicleRoute

Eine `VehicleRoute` hält die vollständige Routenplanung eines einzelnen Fahrzeugs als geordnete Folge von Sublinien.

Feld	Bedeutung
<code>vehicle_id</code>	Nummer des Fahrzeugs
<code>sublines</code>	geordnete Folge der Sublinien, anfangs leer

`sublines` ist bewusst eine *geordnete* Liste: Die Sublinien laufen zeitlich nacheinander ab, und diese Reihenfolge wird für die Wende- und Leerfahrtrechnung zwischen benachbarten Sublinien gebraucht.

Übergänge zwischen Sublinien. Eine Sublinie kennt nur ihre eigene Haltefolge. Der Übergang zum Nachbarn (Leerfahrt und Wende) gehört keiner der beiden Sublinien allein, sondern dem *Paar*: Er ändert sich, sobald sich eine der beiden ändert. Deshalb ist er Information der Routenebene und wird nicht als eigene Datenstruktur (etwa eine „Wende-Aktion“) gespeichert, sondern bei Bedarf berechnet: aus Endstation und Endzeit der vorigen Sublinie, der Startstation der nächsten und der Wendezeit t_{turn} . Der Vorteil ist Redundanzfreiheit: Der Übergang ist durch diese Größen vollständig bestimmt; würde er zusätzlich gespeichert, müsste er nach jeder Einfügung nachgeführt werden und könnte inkonsistent werden. Die Berechnung bei Bedarf ist dagegen ein $\mathcal{O}(1)$ -Zugriff und stets aktuell. Die *Startzeit* der nächsten Sublinie ist dabei kein gespeicherter Wert, sondern ergibt sich genau aus diesem Übergang (Endzeit plus Leerfahrtdauer plus Wendezeit, frühestens der früheste Abholzeitpunkt der ersten Anfrage). Bei jeder Änderung der Route wird sie über die Zeitfortschreibung neu berechnet (Abschnitte 5.1.1 und 5.3.1).

5.2.6 Solution

Die **Solution** fasst alle Fahrzeugrouten sowie die Mengen der noch ungeplanten und bereits abgelehnten Anfragen zusammen.

Feld	Bedeutung
routes	die Routen aller Fahrzeuge
unscheduled	noch einzuplanende Anfragen (Indizes)
rejected	endgültig abgelehnte Anfragen
instance	Verweis auf die zugehörige Probleminstance

Die Trennung von **unscheduled** und **rejected** ist algorithmisch wichtig: Anfragen in **unscheduled** werden in jedem Schritt der Heuristik neu bewertet; eine Anfrage wird erst dann nach **rejected** verschoben, wenn in der gesamten aktuellen Lösung keine zulässige Einfügeposition für sie existiert.

5.2.7 InsertionCandidate

Während der Heuristik werden für jede ungeplante Anfrage alle zulässigen Einfügepositionen als **InsertionCandidate**-Objekte berechnet, sortiert und verglichen.

Feld	Bedeutung
cost	Einfügekosten $c_p(r)$ aus Abschnitt 4.1.4, primärer Sortierschlüssel
pickup_time	Abholzeitpunkt an dieser Position (erster Tie-Break)
subline_rank	Sublinien-Nummer (zweiter Tie-Break; neue Sublinie zuletzt)
vehicle_idx	Fahrzeug, in das eingefügt würde
subline_idx	Ziel-Sublinie (-1 = neue Sublinie anlegen)
position	Index in der Stops-Liste der Ziel-Sublinie, <i>vor</i> den der Pickup kommt
dropoff_pos	Index des Dropoffs in der Stops-Liste (nach dem Pickup-Einschub)
request_idx	Anfrage, um die es geht

Dass bei **cost** kleinere Werte besser sind, folgt aus der Definition der Einfügekosten als negativem Zielfunktionszuwachs (Abschnitt 4.1.4). Es ist keine Konvention des Algorithmus. **position** und **dropoff_pos** sind Indizes in der Stops-Liste der Ziel-Sublinie: **position** bezeichnet die Stelle, vor der der Pickup eingeschoben wird, **dropoff_pos** die Stelle des Dropoffs in der Haltefolge *nach* dem Pickup-Einschub. Beide werden beim tatsächlichen Einfügen gebraucht. Einen Sonderfall bildet **subline_idx** = -1 (neue Sublinie): Dann gibt es noch keine Stops-Liste, und **position** kodiert stattdessen, an welcher Stelle der *Routenfolge* (also zwischen welchen bestehenden Sublinien) die neue Sublinie eingefügt wird; **dropoff_pos** ist in diesem Fall immer 1, weil der Dropoff der zweite Stop der neuen Sublinie ist.

5.2.8 Zusammenfassung der Hierarchie

Die folgende Tabelle fasst die Datenstrukturen und ihre Beziehungen zusammen. Auf der Lösungsseite schachteln sich die Objekte: Eine **Solution** enthält die Routen aller Fahrzeuge, jede **VehicleRoute** ihre Sublinien, jede **Subline** ihre Stops. Auf der Eingabeseite bündelt die **Instance** alle **Request**-Objekte. Beide Seiten sind über das Feld **request_idx** eines **Stop** verbunden, das auf die zugehörige **Request** verweist. Der **InsertionCandidate** steht abseits, weil er keine dauerhafte Struktur ist, sondern nur während der Heuristik eine mögliche Einfügeposition beschreibt.

Klasse	Entspricht	Enthält
Instance	Probleminstanz	Anfragen, Distanzmatrix, Parameter
Request	Anfrage $r \in R$	Zeitfenster, Richtung, Bedienzeit
Solution	Gesamtlösung	VehicleRoute [*] , ungeplante/abgelehnte Anfragen
VehicleRoute	Route eines Fahrzeugs	Subline [*]
Subline	Sublinie s	Stop [*] , Richtung
Stop	geplante Bedienaktion	Station, Zeiten, Last
InsertionCandidate	Einfügungskandidat	Position, Kosten (für Heuristik)

Diese Hierarchie bildet das mathematische Modell aus Kapitel 2 direkt ab und ermöglicht es, alle Zulässigkeitsbedingungen aus Abschnitt 4.1 effizient zu prüfen: Die Richtungs-
lität ist in der Richtung der Sublinie kodiert, die Kapazität im `load_after`-Feld jedes Halts, und die Zeitfenster in den `arrival-/departure`-Feldern der einzelnen Stopps.

5.3 Methoden

Auf den Datenstrukturen des vorigen Abschnitts arbeiten die eigentlichen Berechnungen. Sie laufen in zwei Schritten, die hier auch getrennt dargestellt sind: Zuerst wird bestimmt, *welche* Einfügepositionen gültig sind; danach wird für diese gültigen Positionen der *Mehrweg* (die Einfügekosten) berechnet. Diese Reihenfolge entspricht dem Ablauf im Code. Die Kostenrechnung läuft nur für Positionen, die die Gültigkeitsprüfung bestanden haben.

5.3.1 Gültige Positionen finden

Bevor irgendetwas gerechnet wird, steht die Frage: *Wo* darf die Anfrage überhaupt hin? Kandidatenmenge und Gültigkeitsbedingungen sind in Abschnitt 5.1.5 beschrieben; geprüft wird von billig nach teuer. Richtung und Kapazität sind schnell entschieden, die eigentliche Arbeit steckt in der Verträglichkeit mit dem Rest der Route: Wie in Abschnitt 5.1.1 begründet, wird die gesamte Restroute ab der Einfügestelle als durchgehende Zeitachse nachgerechnet; reißt dabei ein spätestes Zeitfenster oder eine maximale Fahrzeit, ist die Position ungültig.

Die Prüfung übernimmt die Prozedur `PROPAGATEROUTE` (so heißt sie auch im Quellcode: `propagate_route`). Sie führt zwei Laufvariablen mit: `time` und `station`, die aktuelle Zeit und Station des Fahrzeugs entlang der Zeitachse.

Abb. 5.6: Prozedur `PROPAGATEROUTE`: die Restroute als eine durchgehende Zeitachse.

`PROPAGATEROUTE` (Sublinien ab der Einfügestelle):

`time, station` $\leftarrow$ Abfahrtszeit und Station des letzten Halts *vor* der Einfügestelle
(am Routenanfang entfällt beides: Die erste Sublinie beginnt ohne Anfahrt an ihrem ersten Halt,

zum frühesten Abholzeitpunkt ihrer ersten Anfrage; vgl. Abschnitt 5.1.3)

für jede Sublinie in Routenreihenfolge (zeitlich, wie in `sublines` gespeichert):

für jeden Halt der Sublinie:

Ankunft $\leftarrow$ `time` + Fahrzeit von `station` zur Station des Halts

(beim ersten Halt der Sublinie zuzüglich Wendezeit t_{turn} ; ist `station` bereits die Station des Halts, ist die Fahrzeit null; direkte Wende ohne Leerfahrt)

liegt die Ankunft *nach* dem spätesten Zeitpunkt des Halts $\rightarrow$ **ungültig**

Abfahrt $\leftarrow$ $\max(\text{Ankunft, frühester Zeitpunkt des Halts}) + \text{Bedienzeit}$

`time, station` $\leftarrow$ Abfahrt und Station des Halts

überschreitet ein Fahrgast der Sublinie seine maximale Fahrzeit $\rightarrow$ **ungültig**
alle Sublinien bestanden $\rightarrow$ **gültig**

Die Zeitfortschreibung ist also je Halt dieselbe Rechnung: Ankunft aus Abfahrt des Vorgängers plus Fahrzeit, gegebenenfalls warten bis zum frühesten Zeitpunkt, Abfahrt nach der Bedienzeit. Sie bricht ab, sobald ein Halt *nach* seinem spätesten Zeitpunkt läge, und die Fahrzeitprüfung stellt sicher, dass kein Fahrgast länger als seine maximale Fahrzeit unterwegs ist. Besteht die Restroute beide Prüfungen, ist die Position gültig; und *erst dann* wird ihr Mehrweg bestimmt (nächster Abschnitt). So läuft die Kostenrechnung nur für Positionen, die tatsächlich in Frage kommen.

5.3.2 Einfügekosten über die Stop-Folge

Die Einfügekosten einer Anfrage sind schlicht die *Mehrkilometer*, die das Fahrzeug durch sie fährt; und das ist per Definition

$$\text{Kosten} = \text{Strecke}_{\text{neu}} - \text{Strecke}_{\text{alt}}.$$

Statt diese Differenz aus den Fallunterscheidungen aus Abschnitt 5.1.6 samt ihrer Kombinationen zusammenzusetzen, wird sie in *einem* Schritt aus der gesamten Route berechnet. Dazu wird die Fahrzeugroute als *flache Haltefolge* betrachtet: alle Stops aller Sublinien hintereinander. Die gefahrene Strecke ist dann einfach die Summe der Distanzen zwischen aufeinanderfolgenden Stops; und weil zwei benachbarte Stops über eine Sublinien-Grenze hinweg genau die Leerfahrt bilden, stecken die *Leerkilometer automatisch mit drin*.

Zwei schlanke Hilfsstrukturen genügen. Sie werden zusätzlich zu **Subline** und **InsertionCandidate** gebraucht, weil sie eine andere Sicht auf dieselbe Route bieten: **StopRoute** hält die flache Haltefolge ohne Sublinienstruktur (nur auf ihr ist die Gesamtstrecke eine einfache Summe über aufeinanderfolgende Stationen), und **InsertionPosition** beschreibt die *Eingabe* der Kostenrechnung (wohin Pickup und Dropoff kommen), während der **InsertionCandidate** aus Abschnitt 5.2.7 deren bewertetes *Ergebnis* ist.

Eine **InsertionPosition** besteht aus drei Werten: dem Index, *vor* dem der Pickup eingefügt wird, dem Index für den Dropoff und der Ziel-Sublinie (-1 = neue Sublinie). Eine **StopRoute** ist nichts weiter als die Liste aller angefahrenen Stationen der Route hintereinander. Ihre Strecke ist die Summe der Distanzen zwischen je zwei aufeinanderfolgenden Stationen.

Die Einfügekosten ergeben sich damit in drei Schritten: die alte Haltefolge kopieren, Pickup und Dropoff an ihren Indizes einschieben, und die Streckendifferenz $\text{Strecke}_{\text{neu}} - \text{Strecke}_{\text{alt}}$ zurückgeben.

Eine Formel für alle Fälle. Der Mehrwert der Streckendifferenz liegt darin, dass sie sämtliche in Abschnitt 5.1.6 einzeln diskutierten Situationen automatisch korrekt abdeckt; nichts davon muss gesondert behandelt werden:

- *Anfrage liegt auf dem Weg*; kostet 0: Liegt der neue Halt zwischen zwei bereits angefahrenen Stationen (etwa Station 3 zwischen den Stationen 1 und 5), gilt auf der Linie die Dreiecksungleichheit $t_{1,5} = t_{1,3} + t_{3,5}$; der Zwischenstopp fügt der Summe keine Strecke hinzu.
- *Verlängerung* vorne oder hinten (Fälle 1 und 2 aus Abschnitt 5.1.6); das zusätzliche Teilstück steckt in der Summe.
- *Hin-und-Her beim Wenden*; fährt die Route etwa von Station 5 hinauf nach 7 und zurück nach 3, zählt die Summe beide Teilstücke; die doppelte Durchfahrt aus Abschnitt 5.1.6 ergibt sich von selbst.
- *Leerfahrten* zwischen Sublinien, als Distanz zwischen zwei aufeinanderfolgenden Stops mitgezählt.
- *Neue Sublinie* eröffnen; dieselbe Formel, nur mit anderen Einfügepositionen.

Die Fallunterscheidungen aus Abschnitt 5.1.6 samt ihrer Kombinationen müssen also nicht einzeln ausprogrammiert werden. Die Streckendifferenz erfasst sie alle.

Laufzeit. Der Preis dieser Ein-Formel-Lösung gegenüber einer lokalen Fallunterscheidung ist ein etwas höherer Rechenaufwand: Die Kostenrechnung baut die neue Haltefolge

auf und summiert die *gesamte* Route ($O(n)$ bei n Stops), statt nur die drei an der Einfügestelle betroffenen Distanzen anzufassen ($O(1)$). Bei den Instanzgrößen dieser Arbeit fällt das nicht ins Gewicht, denn auch die Gültigkeitsprüfung aus Abschnitt 5.3.1 läuft in $O(n)$ über die Restroute. Die Kostenrechnung erhöht die Größenordnung des Gesamtaufwands also nicht. Dafür ist die Rechnung *fehlerfrei*: Es gibt keine Fallunterscheidung, in der sich ein Vorzeichen- oder Grenzfehler verstecken könnte. Wo Geschwindigkeit einmal zählt, liefert die lokale $O(1)$ -Formel $t_{\text{davor}, \text{neu}} + t_{\text{neu}, \text{danach}} - t_{\text{davor}, \text{danach}}$ exakt denselben Wert. Dabei ist t die Fahrzeitmatrix aus Abschnitt 2.4, und *davor*, *neu*, *danach* bezeichnen die Station unmittelbar vor der Einfügestelle, die neu eingefügte und die unmittelbar danach.

Beste Position. Für jedes Fahrzeug werden alle gültigen Positionen so bewertet. Die mit den kleinsten Kosten ist die beste Einfügeposition dieses Fahrzeugs. Diese Fahrzeug-Bestwerte gehen anschließend in die Regret- k -Auswahl ein (Abschnitt 5.1.8).

6 Evaluation

6.1 Versuchsaufbau

Die Heuristiken werden auf den 14 *Basic*-Benchmark-Instanzen von Reiter et al. [RSS24] ausgewertet (2 bis 5 Fahrzeuge, 16 bis 58 Anfragen, Kapazität 3, reale Distanzmatrix der Würzburger Buslinie 6); jede Anfrage umfasst dort genau einen Fahrgast ($q_r = 1$), sodass Akzeptanzrate und beförderte Fahrgäste zusammenfallen. Verglichen werden die *Greedy*-Insertion als Baseline und die *Regret- k* -Insertion für $k = 2, \dots, \kappa$ (Fahrzeugzahl κ). Zielfunktion ist $w_1 \cdot (\text{beförderte Fahrgäste}) + w_2 \cdot (\text{eingesparte Strecke})$ mit $w_1 = 10$ und $w_2 = 1$. Die *customer-focused*-Gewichtung aus [RSS24], die die Akzeptanz stark priorisiert und den direkten Vergleich mit den dortigen Referenzläufen erlaubt. Alle Verfahren sind deterministisch; einzig das **random**-Seeding zieht seine Startanfragen zufällig. Das Seeding (Abschnitt 5.1.4) gehört dabei ausschließlich zur *Regret- k* -Insertion: Es bricht die anfängliche Symmetrie der leeren Fahrzeuge auf, ohne die der *Regret*-Wert zu Beginn keine Anfrage unterscheiden könnte. Für *Regret- k* werden die vier expliziten Seeding-Strategien gegeneinander verglichen (Abschnitt 6.2); als einfacher Vergleichswert läuft zusätzlich ein **random**-Seeding mit, das die κ Startanfragen zufällig zieht (ein einzelner Lauf). Die beste Strategie wird für den Vergleich mit dem exakten Verfahren verwendet. Die Bedienzeit b_r fällt an jedem Halt an (Ein- wie Ausstieg, vgl. Abschnitt 2.4) und beträgt in den Benchmark-Instanzen einheitlich 3 Minuten. Die Wendezeit beträgt (wie in den Referenzläufen des exakten Verfahrens) null.

Jede erzeugte Lösung wurde mit einem *unabhängigen* Zulässigkeitsprüfer nachgerechnet, der Kapazität, Zeitfenster und maximale Fahrtzeit direkt aus den Instanzdaten prüft. *Keine* der Lösungen enthält eine Verletzung. Die berichteten Werte sind also durchweg zulässig. Der in Abschnitt 5.1.7 beschriebene Randfall einer zulässigen, aber wertsenkenden Einfügung trat in den Experimenten praktisch nicht auf: Derart große Umwege scheitern bei den Zeitfenstern der Benchmark-Instanzen meist schon an der Zulässigkeit.

Die Evaluation gliedert sich in fünf Experimente: Abschnitt 6.2 vergleicht die vier Seeding-Strategien für *Regret- k* gegen die Greedy-Baseline und wählt die beste aus. Abschnitt 6.3 berechnet mit dem ereignisbasierten MILP exakte Referenzlösungen für alle Basis-Instanzen. Abschnitt 6.4 misst daran den Optimality Gap der Heuristik, Abschnitt 6.5 wiederholt Heuristik- und MILP-Läufe auf den 19 erweiterten Instanzen, und Abschnitt 6.6 untersucht den Einfluss der Gewichtsparameter. Alle Experimente liefen auf demselben Rechner (Intel Core i7-12650H, 16 GB RAM, Linux). Die angegebene Rechenzeit der Heuristik ist der Mittelwert aus drei Läufen und dient nur dem Größenvergleich, nicht als absolute Laufzeitmessung.

6.2 Ergebnisse und Wahl der Seeding-Strategie

Als Bezugspunkt dient zunächst die Greedy-Insertion (Tabelle 6.1). Sie zeigt das in Kapitel 4 beschriebene Problem unmittelbar: Auf den engen Zwei-Fahrzeug-Instanzen bleiben Anfragen unbedient (w2-16: drei von 16, w2-20: sechs von 20, w3-24: zwei von 24), weil frühe billige Einfügungen späteren Anfragen die letzten zulässigen Positionen nehmen.

Tab. 6.1: Greedy-Insertion als Baseline auf den Basic-Instanzen. km = gefahrene Kilometer; $Direkt$ = Summe der Direktstrecken; Δ = Direkt – km .

Instanz	κ	Zielwert	Akzeptanz [%]	km	Direkt	Δ
w2-16	2	115	81,3	75	68	–7
w4-16	4	145	100	69	54	–15
w3-18	3	166	100	76	62	–14
w2-20	2	116	70,0	62	63	+1
w2-24	2	198	100	139	97	–42
w3-24	3	205	91,7	89	81	–8
w4-24	4	221	100	87	68	–19
w3-30	3	268	100	128	96	–32
w4-32	4	283	100	125	88	–37
w3-36	3	341	100	141	122	–19
w4-40	4	365	100	177	142	–35
w5-40	5	367	100	162	129	–33
w5-50	5	519	100	178	197	+19
w4-58	4	560	100	253	233	–20

Für die Regret- k -Insertion wird jedem Fahrzeug vorab eine Startanfrage (*Seed*) zugewiesen, die seine erste Sublinie eröffnet (Abschnitt 5.1.4). Vier Auswahlkriterien werden verglichen: **zeit** (frühester Abfahrtszeitpunkt), **laenge** (längste Direktstrecke), **isoliert** (größte räumliche Isolation, nach dem Dezentralisierungsgedanken von Diana und Des-souky [DD04]) und **richtung** (je zur Hälfte auf- und absteigende Seeds). Die Tabellen 6.2 bis 6.5 zeigen die Zielwerte je Strategie für alle Regret- k mit $k = 2, \dots, \kappa$; „–“ bedeutet, dass k die Fahrzeugzahl κ der Instanz übersteigt. Die Spalte *Best* ist der *beste erreichte* Zielwert über die Regret-Varianten. Die Auswahl der besten Strategie stützt sich ausschließlich auf diesen Bestwert. Die Spalten km , *Direkt* und Δ sind wie in Tabelle 6.1 definiert und beziehen sich auf die beste Variante. Beim Vergleich der km -Werte ist zu beachten, dass nicht jede beste Lösung alle Anfragen bedient: Bei **zeit** und **richtung** bleiben auf w2-16, w2-20 und w2-24 insgesamt fünf Anfragen unbedient (bei **zeit** zudem eine auf w3-36, bei **richtung** eine auf w3-30), bei **isoliert** je eine auf w2-20 und w2-24, bei **laenge** einzig eine auf w2-20; weniger bediente Anfragen bedeuten tendenziell auch weniger gefahrene Kilometer.

Abbildung 6.1 stellt die Bestwerte aller Strategien über die Instanzen direkt gegenüber. Sichtbar wird beides auf einen Blick: Die Kurven liegen eng beieinander (die Wahl der Strategie macht wenig aus), und **laenge** verläuft überwiegend am oberen Rand, während die Greedy-Baseline auf den engen Zwei-Fahrzeug-Instanzen (w2-16, w2-20) deutlich

Tab. 6.2: Seeding *zeit* (frühester Abfahrtszeitpunkt). Angegeben sind Zielfunktionswerte; „–“ = k übersteigt die Fahrzeugzahl. *Best* = bester Zielwert über die Regret-Varianten (je Instanz fett gedruckt); *km*, *Direkt*, Δ wie in Tabelle 6.1.

Instanz	κ	Reg-2	Reg-3	Reg-4	Reg-5	Best	km	Direkt	Δ
w2-16	2	125	–	–	–	125	92	68	–24
w4-16	4	144	146	138	–	146	68	54	–14
w3-18	3	161	163	–	–	163	79	62	–17
w2-20	2	153	–	–	–	153	86	63	–23
w2-24	2	191	–	–	–	191	119	97	–22
w3-24	3	202	214	–	–	214	107	81	–26
w4-24	4	218	217	220	–	220	88	68	–20
w3-30	3	262	249	–	–	262	134	96	–38
w4-32	4	294	296	292	–	296	112	88	–24
w3-36	3	285	311	–	–	311	156	122	–34
w4-40	4	394	384	390	–	394	148	142	–6
w5-40	5	361	374	376	370	376	153	129	–24
w5-50	5	502	499	497	495	502	195	197	+2
w4-58	4	500	569	525	–	569	244	233	–11

einbricht.

Tabelle 6.6 fasst den Vergleich über alle Strategien zusammen.

Auswahl. *laenge* erzielt die höchste Summe der Bestwerte (4056), mit Abstand vor *isoliert* (4016) und den beiden zeit- bzw. richtungsbasierten Strategien (3922/3955). Alle vier liegen über der Greedy-Baseline (3869). Das Bild bestätigt sich auch instanzweise: *laenge* liefert auf neun der 14 Instanzen den besten Zielwert aller Strategien (einmal gemeinsam mit *isoliert*), *isoliert* auf fünf, *richtung* auf einer und *zeit* auf keiner. Die Summe ist hier also kein Artefakt einzelner Ausreißer. Bemerkenswert ist zudem das zufällige Seeding: Mit 4005 ordnet es sich mitten zwischen den Strategien ein und schlägt *zeit* wie *richtung*. Die Strategiewahl macht also wenig aus, solange die Startanfragen gestreut werden; nur *laenge* hebt sich ab. Das passt zur Erwartung aus Abschnitt 5.1.4: Lange Startanfragen eröffnen große Sublinien, in die sich viele kürzere, überlappende Anfragen nahezu ohne Mehrweg einfügen lassen; und lange Anfragen tragen selbst am meisten zur eingesparten Distanz bei. Alle folgenden Auswertungen verwenden daher die *laenge*-Strategie.

Auffällig ist der deutliche Abstand von *zeit* und *richtung* (3922/3955) nach unten: Eine ungünstige Startwahl kann die Lösung aktiv verschlechtern, vermutlich weil die zeitlich frühesten Anfragen die Fahrzeuge in denselben Linienabschnitten häufen (vgl. Abschnitt 5.1.4).

Ergebnisse der gewählten Strategie. Mit *laenge*-Seeding übertrifft Regret- k die Greedy-Baseline auf zwölf der 14 Instanzen, teils deutlich (w2-20: 164 gegenüber 116; w2-16: 131 gegenüber 115). Die großen Abstände entstehen dort, wo Greedy Anfragen ablehnt, die Regret- k noch unterbringt: Auf w2-16 bedient Regret- k alle 16 Anfragen, Greedy nur 13. Auf w2-20 sind es 19 gegenüber 14, auf w3-24 alle 24 gegenüber 22.

Tab. 6.3: Seeding **laenge** (längste Direktstrecke). Angegeben sind Zielfunktionswerte; „-“ = k übersteigt die Fahrzeugzahl. *Best* = bester Zielwert über die Regret-Varianten (je Instanz fett gedruckt); *km*, *Direkt*, Δ wie in Tabelle 6.1.

Instanz	κ	Reg-2	Reg-3	Reg-4	Reg-5	Best	km	Direkt	Δ
w2-16	2	131	–	–	–	131	97	68	–29
w4-16	4	145	144	147	–	147	67	54	–13
w3-18	3	168	165	–	–	168	74	62	–12
w2-20	2	164	–	–	–	164	87	63	–24
w2-24	2	208	–	–	–	208	129	97	–32
w3-24	3	218	217	–	–	218	103	81	–22
w4-24	4	229	224	218	–	229	79	68	–11
w3-30	3	270	280	–	–	280	116	96	–20
w4-32	4	288	286	286	–	288	120	88	–32
w3-36	3	329	334	–	–	334	148	122	–26
w4-40	4	406	410	392	–	410	132	142	+10
w5-40	5	377	372	377	377	377	152	129	–23
w5-50	5	510	500	503	497	510	187	197	+10
w4-58	4	592	582	562	–	592	221	233	+12

Jeder zusätzlich beförderte Fahrgast bringt $w_1 = 10$ Punkte, sodass sich eine höhere Akzeptanz stark im Zielwert niederschlägt. Wo bereits Greedy alle Fahrgäste bedient, kann Regret- k nur noch die gefahrene Strecke optimieren. Dieser Vorteil ist mit $w_2 = 1$ schwach gewichtet und kann sich umkehren. Auf w3-36 (341 gegenüber 334) und w5-50 (519 gegenüber 510) liegt die Baseline vorn. Der Nutzen von Regret- k hängt damit weniger an der Instanzgröße als am *Annahme-Druck*: daran, ob überhaupt Anfragen abgelehnt zu werden drohen.

Zwischen den einzelnen k -Werten ergibt sich kein einheitliches Bild: Ein höheres k verbessert den Bestwert nur vereinzelt (unter **laenge** etwa w4-16 mit 147 bei $k = 4$), häufig fällt der Zielwert mit wachsendem k sogar ab (etwa w4-40: 410 bei $k = 3$, 392 bei $k = 4$). Ein größeres k lohnt sich also nicht systematisch.

Die Rechenzeiten zeigt Tabelle 6.7: Sie wachsen mit der Instanzgröße von rund 10 Millisekunden (w4-16) auf gut 600 Millisekunden (w4-58) und unterscheiden sich zwischen den Seeding-Strategien nur unwesentlich. Das Seeding selbst fällt mit einmaligem Sortieren der Anfragen nicht ins Gewicht. Für Greedy und Regret- k sind die Zeiten vergleichbar, da beide in jedem Schritt alle Einfügepositionen jeder offenen Anfrage bewerten.

Tab. 6.4: Seeding *isoliert* (größte räumliche Isolation). Angegeben sind Zielfunktionswerte; „–“ = k übersteigt die Fahrzeugzahl. *Best* = bester Zielwert über die Regret-Varianten (je Instanz fett gedruckt); *km*, *Direkt*, Δ wie in Tabelle 6.1.

Instanz	κ	Reg-2	Reg-3	Reg-4	Reg-5	Best	km	Direkt	Δ
w2-16	2	131	–	–	–	131	97	68	–29
w4-16	4	145	148	143	–	148	66	54	–12
w3-18	3	164	166	–	–	166	76	62	–14
w2-20	2	160	–	–	–	160	91	63	–28
w2-24	2	185	–	–	–	185	141	97	–44
w3-24	3	217	206	–	–	217	104	81	–23
w4-24	4	225	218	222	–	225	83	68	–15
w3-30	3	266	265	–	–	266	130	96	–34
w4-32	4	291	297	292	–	297	111	88	–23
w3-36	3	334	342	–	–	342	140	122	–18
w4-40	4	400	400	404	–	404	138	142	+4
w5-40	5	372	370	377	375	377	152	129	–23
w5-50	5	517	492	501	473	517	180	197	+17
w4-58	4	581	565	544	–	581	232	233	+1

Tab. 6.5: Seeding *richtung* (je zur Hälfte auf-/absteigend). Angegeben sind Zielfunktionswerte; „–“ = k übersteigt die Fahrzeugzahl. *Best* = bester Zielwert über die Regret-Varianten (je Instanz fett gedruckt); *km*, *Direkt*, Δ wie in Tabelle 6.1.

Instanz	κ	Reg-2	Reg-3	Reg-4	Reg-5	Best	km	Direkt	Δ
w2-16	2	125	–	–	–	125	92	68	–24
w4-16	4	144	146	138	–	146	68	54	–14
w3-18	3	163	163	–	–	163	79	62	–17
w2-20	2	153	–	–	–	153	86	63	–23
w2-24	2	191	–	–	–	191	119	97	–22
w3-24	3	216	206	–	–	216	105	81	–24
w4-24	4	222	223	200	–	223	85	68	–17
w3-30	3	262	263	–	–	263	119	96	–23
w4-32	4	296	294	285	–	296	112	88	–24
w3-36	3	329	296	–	–	329	153	122	–31
w4-40	4	394	384	390	–	394	148	142	–6
w5-40	5	361	371	378	370	378	151	129	–22
w5-50	5	504	498	509	502	509	188	197	+9
w4-58	4	500	569	525	–	569	244	233	–11

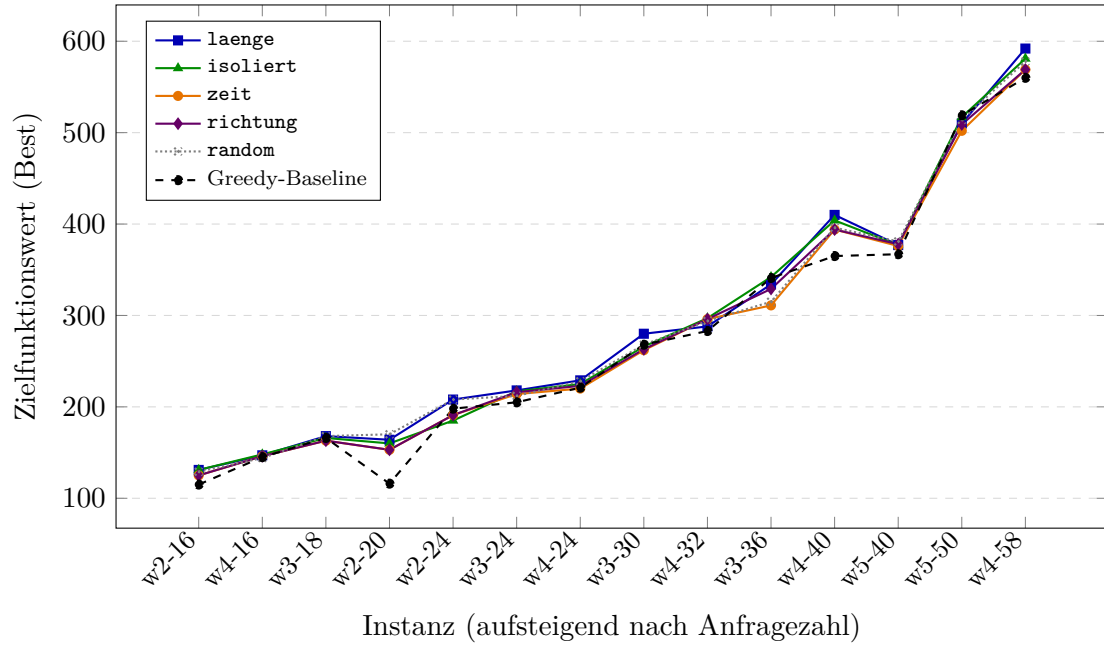


Abb. 6.1: Beste Zielfunktionswerte je Seeding-Strategie über die 14 Basis-Instanzen (Werte aus den Tabellen 6.2 bis 6.5), dazu **random** (ein Lauf) und die Greedy-Baseline.

Tab. 6.6: Vergleich der Seeding-Strategien. $\sum \text{Best}$ = Summe der Bestwerte (beste Regret-Variante) über alle 14 Instanzen; Besser/Gleich/Schlechter = Anzahl der Instanzen, auf denen die beste Regret-Variante besser, gleich bzw. schlechter abschneidet als die Greedy-Baseline. **random** ist ein einzelner Lauf mit zufällig gezogenen Startanfragen; sortiert nach $\sum \text{Best}$.

Strategie	$\sum \text{Best}$	Regret vs. Greedy		
		Besser	Gleich	Schlechter
laenge	4056	12	0	2
isoliert	4016	10	1	3
random	4005	10	1	3
richtung	3955	9	0	5
zeit	3922	8	0	6
Greedy-Baseline	3869	—		

Tab. 6.7: Mittlere Laufzeit der Regret-2-Insertion in Millisekunden je Seeding-Strategie (Mittel aus drei Läufen; Python, Einzelkern).

Instanz	zeit	laenge	isoliert	richtung
w2-16	12	14	14	11
w4-16	11	11	11	11
w3-18	13	16	15	14
w2-20	20	28	25	22
w2-24	42	43	42	41
w3-24	35	37	39	44
w4-24	38	33	36	38
w3-30	84	97	82	77
w4-32	94	96	104	91
w3-36	149	145	139	144
w4-40	179	174	171	179
w5-40	198	157	185	180
w5-50	337	306	271	327
w4-58	663	547	553	667

6.3 Exakte Referenzlösungen (Event-Based-MILP)

Als Grundlage für den Vergleich wurden die Instanzen zusätzlich mit dem exakten Verfahren gelöst: der ereignisbasierten MILP-Formulierung des liDARP von Reiter et al. [RSS24], die dort als schnellste der drei vorgestellten Formulierungen ausgewiesen ist. Verwendet wurde deren Solver-Implementierung (Julia 1.11.6, Gurobi 13.0.2 über Gurobi.jl 1.9.2) mit derselben Zielfunktion wie die Heuristik ($w_1 = 10$, $w_2 = 1$) und einem Zeitlimit von 1800 Sekunden je Instanz. Heuristik und exaktes Verfahren verwenden dieselbe reale Distanzmatrix der Würzburger Buslinie 6, sodass die Zielwerte direkt vergleichbar sind.

Anzumerken ist, dass Reiter et al. [RSS24] für dieselbe Formulierung deutlich kürzere Laufzeiten berichten (alle Basic-Instanzen in unter einer Sekunde). Die dort verwendete Implementierung ist jedoch eine andere: Das begleitende Repository zum Paper setzt das ereignisbasierte Modell in C++ mit CPLEX um, während die hier eingesetzte Implementierung in Julia mit Gurobi arbeitet. Die folgenden Werte sind daher als eigene Messung unter der hier beschriebenen Konfiguration zu lesen. Die Laufzeiten beider Implementierungen sind nicht direkt vergleichbar. Erreicht der Solver das Zeitlimit, werden die beste gefundene Lösung (*Incumbent*) und die bewiesene obere Schranke ausgewiesen. Die Ausgaberoutine des Solvers wurde dafür minimal erweitert, da sie im Original bei Zeitlimit keine Lösungswerte ausgibt.

Lösungswerte. Tabelle 6.8 fasst die exakten Ergebnisse zusammen. Für alle Instanzen mit bis zu 32 Anfragen weist der Solver das Optimum nach (w4-32 nach 908 Sekunden); in jeder Lösung werden sämtliche Anfragen bedient (Akzeptanz 100 %), was der *customer-focused*-Gewichtung $w_1 = 10$ entspricht. Ab 36 Anfragen erreicht der Solver das Zeitlimit, liefert aber Incumbents mit bewiesenen MIP-Gaps von höchstens 2,6 %. Die gefundenen Lösungen sind also nahezu optimal, nur der Optimalitätsnachweis steht aus.

Für die Instanzen mit bis zu 32 Anfragen liefert das exakte Verfahren damit bewiesene Optima als Vergleichswerte. Für die fünf größeren liefert es Incumbents mit bewiesenen oberen Schranken. Der Optimalitätsnachweis skaliert jedoch nicht: Schon w4-32 benötigt gut 15 Minuten, ab 36 Anfragen reicht die halbe Stunde nicht mehr. Die Ursache liegt im steilen Wachstum des Ereignisgraphen (von gut 1000 Variablen bei 16 Anfragen auf über 28 000 bei 58 Anfragen), wobei der Aufwand fast vollständig im Lösen steckt, nicht im Modellaufbau. Genau in diesem Größenbereich zeigt sich der praktische Nutzen der Heuristik, die auch für die größten Instanzen in weniger als einer Sekunde eine zulässige Lösung liefert.

Tab. 6.8: Exakte Lösungswerte der ereignisbasierten MILP-Formulierung auf den Basic-Instanzen (reale Würzburg-Distanzmatrix, $w_1 = 10$, $w_2 = 1$, Zeitlimit 1800 s). Bei „Zeitlimit“ ist *obj* die beste gefundene Lösung (Incumbent) und *Schranke* die bewiesene obere Schranke; der Gap ist ihr relativer Abstand.

Instanz	$ R $	κ	Status	obj	Schranke	Gap [%]	Akzeptanz [%]	gespart	Distanz	Zeit [s]
w2-16	16	2	optimal	178	178	0,0	100	18	50	3,4
w4-16	16	4	optimal	165	165	0,0	100	5	49	0,2
w3-18	18	3	optimal	196	196	0,0	100	16	46	0,4
w2-20	20	2	optimal	210	210	0,0	100	10	53	0,7
w2-24	24	2	optimal	268	268	0,0	100	28	69	8,0
w3-24	24	3	optimal	265	265	0,0	100	25	56	9,1
w4-24	24	4	optimal	260	260	0,0	100	20	48	16,9
w3-30	30	3	optimal	321	321	0,0	100	21	75	32,6
w4-32	32	4	optimal	345	345	0,0	100	25	63	908
w3-36	36	3	Zeitlimit	392	401,0	2,3	100	32	90	1800
w4-40	40	4	Zeitlimit	449	455,3	1,4	100	49	93	1800
w5-40	40	5	Zeitlimit	445	454,3	2,1	100	45	84	1800
w5-50	50	5	Zeitlimit	576	590,3	2,5	100	76	121	1800
w4-58	58	4	Zeitlimit	676	693,2	2,6	100	96	137	1800

6.4 Vergleich

Da Heuristik und exaktes Verfahren dieselbe Distanzmatrix und dieselbe Zielfunktion verwenden, sind ihre Zielwerte direkt vergleichbar. Tabelle 6.9 führt dazu die Ergebnisse aus Tabelle 6.3 und Tabelle 6.8 zusammen: Die Spalte *Heuristik* ist die Best-Spalte aus Tabelle 6.3 (bester Zielwert über die Regret-Varianten mit **1aenge**-Seeding), die Spalte *MILP (opt.)* das nachgewiesene Optimum aus Tabelle 6.8. Der *Optimality Gap* ist

$$\text{Gap} = \frac{\text{obj}_{\text{MILP}} - \text{obj}_{\text{Heur.}}}{\text{obj}_{\text{MILP}}}. \quad (6.1)$$

Die Kilometerspalten sind bei voller Akzeptanz aus den Zielwerten ableitbar: Wegen des Zielfunktionsaufbaus $\text{obj} = w_1 \cdot m + (\text{Direkt} - \text{km})$ gilt dann $\text{km} = \text{Direkt} + 10 \cdot |R| - \text{obj}$; für das MILP entspricht das der Differenz aus Direktsumme und der in Tabelle 6.8 ausgewiesenen Ersparnis. Auf w2-20, wo die Heuristik eine Anfrage ablehnt, ist der km-Wert direkt aus der Lösung gemessen.

Tab. 6.9: Optimality Gap der Heuristik (**1aenge**-Seeding, beste Regret-Variante) gegenüber dem MILP-Optimum auf den neun exakt gelösten Instanzen. *Akz.* = Akzeptanzrate der Heuristik (das MILP bedient überall alle Anfragen); *Direkt* = Summe der Direktstrecken aller Anfragen; *km* = tatsächlich gefahrene Kilometer.

Instanz	R	Heuristik	MILP (opt.)	Gap [%]	Akz. [%]	Direkt	km Heur.	km MILP
w2-16	16	131	178	26,4	100	68	97	50
w4-16	16	147	165	10,9	100	54	67	49
w3-18	18	168	196	14,3	100	62	74	46
w2-20	20	164	210	21,9	95	63	87	53
w2-24	24	208	268	22,4	100	97	129	69
w3-24	24	218	265	17,7	100	81	103	56
w4-24	24	229	260	11,9	100	68	79	48
w3-30	30	280	321	12,8	100	96	116	75
w4-32	32	288	345	16,5	100	88	120	63
<i>Mittlerer Gap</i>				17,2				

Die Heuristik erreicht auf acht der neun Instanzen die *optimale Akzeptanz* (100%, wie das MILP); einzig auf w2-20 bleibt eine der 20 Anfragen unbedient. Der Abstand entsteht damit fast ausschließlich aus der Routeneffizienz, also der gesparten Distanz. Die Kilometerspalten machen das sichtbar: Das MILP fährt auf jeder der neun Instanzen *weniger* als die Summe der Direktstrecken, spart also durch Bündelung tatsächlich Kilometer ein. Die Heuristik fährt dagegen auf allen neun Instanzen *mehr* als die Direktsumme (etwa w2-16: 97 km gegenüber 68 km Direktstrecke, während das MILP mit 50 km auskommt). Ihre Ersparnis im Zielfunktionssinn ist hier negativ, und der volle Zielwert stammt aus der Akzeptanz. Der mittlere Gap beträgt 17,2 % und schrumpft mit steigender Fahrzeugzahl (von durchschnittlich rund 24 % bei zwei Fahrzeugen auf rund 13 % bei vier), weil mehr Fahrzeuge mehr Einfügefähigkeit bieten.

Für die fünf größeren Instanzen ist das Optimum nicht bewiesen. Dort tritt an seine Stelle die bewiesene *obere Schranke* aus Tabelle 6.8. Der Gap gegen die Schranke ist *konservativ*: Das wahre Optimum liegt zwischen Incumbent und Schranke, der tatsächliche Abstand der Heuristik zum Optimum ist also höchstens so groß wie der ausgewiesene Wert. Tabelle 6.10 zeigt das Ergebnis.

Tab. 6.10: Konservativer Gap der Heuristik (*laenge*-Seeding) gegenüber der bewiesenen oberen Schranke auf den fünf Instanzen ohne Optimalitätsnachweis. *Akz.*, *Direkt* und *km* wie in Tabelle 6.9; die MILP-Kilometer beziehen sich auf den Incumbent.

Instanz	$ R $	Heuristik	MILP-Inc.	Schranke	Gap [%]	Akz. [%]	Direkt	km Heur.	km MILP
w3-36	36	334	392	401,0	16,7	100	122	148	90
w4-40	40	410	449	455,3	9,9	100	142	132	93
w5-40	40	377	445	454,3	17,0	100	129	152	84
w5-50	50	510	576	590,3	13,6	100	197	187	121
w4-58	58	592	676	693,2	14,6	100	233	221	137
<i>Mittlerer Gap</i>					14,4				

Auch auf den großen Instanzen bleibt der Abstand damit in derselben Größenordnung wie auf den exakt gelösten (10–17 %, im Mittel 14,4 %). Die Lösungsqualität der Heuristik bricht mit wachsender Instanzgröße also nicht ein. Auffällig ist das Kilometerbild: Auf drei der fünf großen Instanzen (w4-40, w5-50, w4-58) unterschreitet auch die Heuristik die Direktsumme, erzielt dort also selbst eine positive Ersparnis; offenbar bieten mehr Anfragen pro Fahrzeug genug Bündelungsgelegenheiten, dass sich die Einfügungen gegenseitig tragen. Zusammen ergibt sich damit für alle 14 Benchmark-Instanzen eine belastbare Gütegarantie, während die Heuristik jede Instanz in weniger als einer Sekunde löst. Die Wahl der *laenge*-Strategie bestätigt sich auch an diesen Gaps: Die zweitbeste Strategie (*isoliert*) läge im Mittel bei 18,8 % bzw. 14,3 %.

Abbildung 6.2 (linker Block) stellt die drei Kilometerwerte aus den Tabellen 6.9 und 6.10 über alle 14 Instanzen dar, aufsteigend sortiert nach der Direktsumme. Zwei Trends werden sichtbar: Das MILP fährt auf jeder Instanz weniger als die Direktsumme, und sein Vorsprung wächst mit der Instanzgröße (von rund 10 % Ersparnis auf w4-16 bis über 40 % auf w4-58). Die Heuristik liegt auf den kleinen und mittleren Instanzen über der Direktsumme, nähert sich ihr mit wachsender Größe aber an und unterschreitet sie ab w4-40. Der Kilometerabstand zwischen Heuristik und MILP wächst absolut zwar mit der Instanzgröße, gemessen an der Direktsumme nimmt er auf den größten Instanzen jedoch ab: Auf w2-16 beträgt die Mehrfahrt der Heuristik gegenüber dem MILP 47 km bei nur 68 km Direktsumme, auf w4-58 sind es 84 km bei 233 km. Relativ betrachtet wird der Rückstand bei längeren Gesamtstrecken also eher kleiner. Das passt zum niedrigeren mittleren Gap auf den großen Instanzen.

6.5 Erweiterte Instanzen

Zusätzlich zu den 14 Basis-Instanzen wurde die Heuristik auf den 19 erweiterten Benchmark-Instanzen ausgeführt (48 bis 132 Anfragen, 5 bis 11 Fahrzeuge), mit demselben Versuchsaufbau wie in Abschnitt 6.1 und **laenge**-Seeding. Tabelle 6.11 zeigt die Ergebnisse: Die Heuristik bedient auf allen 19 Instanzen sämtliche Anfragen, und die Lösungsvalidierung meldet keine Nebenbedingungsverletzungen. Die Greedy-Baseline (Spalte *Greedy*) bleibt meist knapp dahinter und lässt auf w6-72 eine Anfrage unbedient. Auf drei Instanzen (w8-80, w9-108, w11-110) liegt sie allerdings vor der besten Regret-Variante.

Tab. 6.11: Heuristik (**laenge**-Seeding) auf den 19 erweiterten Instanzen. *Variante* = Regret-Ordnung, die den Bestwert erreicht; *Greedy* = Zielwert der Greedy-Baseline; *km*, *Direkt* und Δ wie in Abschnitt 6.2. Die Akzeptanz der besten Regret-Variante beträgt auf allen Instanzen 100%; die Greedy-Baseline lässt auf w6-72 eine Anfrage unbedient.

Instanz	$ R $	κ	Best	Variante	Greedy	km	Direkt	Δ
w6-48	48	6	481	Reg-5	480	183	184	+1
w7-56	56	7	554	Reg-5	552	183	177	−6
w5-60	60	5	588	Reg-2	583	220	208	−12
w6-60	60	6	590	Reg-4	568	210	200	−10
w8-64	64	8	652	Reg-4	647	208	220	+12
w7-70	70	7	690	Reg-3	688	244	234	−10
w6-72	72	6	718	Reg-4	700	279	277	−2
w9-72	72	9	746	Reg-9	732	242	268	+26
w8-80	80	8	832	Reg-4	845	269	301	+32
w10-80	80	10	805	Reg-10	789	283	288	+5
w7-84	84	7	831	Reg-2	811	278	269	−9
w11-88	88	11	885	Reg-2	879	282	287	+5
w9-90	90	9	928	Reg-7	923	290	318	+28
w8-96	96	8	995	Reg-7	969	308	343	+35
w10-100	100	10	1057	Reg-2	1048	312	369	+57
w9-108	108	9	1104	Reg-7	1108	339	363	+24
w11-110	110	11	1140	Reg-3	1141	334	374	+40
w10-120	120	10	1235	Reg-3	1219	339	374	+35
w11-132	132	11	1363	Reg-10	1339	405	448	+43

Der auf den Basis-Instanzen beobachtete Kilometerrend setzt sich fort und verstärkt sich: Δ ist auf 13 der 19 Instanzen positiv, ab 88 Anfragen ausnahmslos, mit bis zu +57 km auf w10-100 (312 km gefahren bei 369 km Direktsumme). Der rechte Block von Abbildung 6.2 zeigt beide Kurven für die erweiterten Instanzen, wieder aufsteigend nach der Direktsumme sortiert: Die Heuristik-Kurve löst sich mit wachsender Instanzgröße sichtbar von der Direktsumme nach unten. Auffällig ist außerdem, dass auf den großen Flotten häufig hohe Regret-Ordnungen den Bestwert liefern (Reg-7 bis Reg-10 auf sechs Instanzen). Bei vielen Fahrzeugen lohnt sich also der Blick auf mehr als die zwei besten Einfügeoptionen.

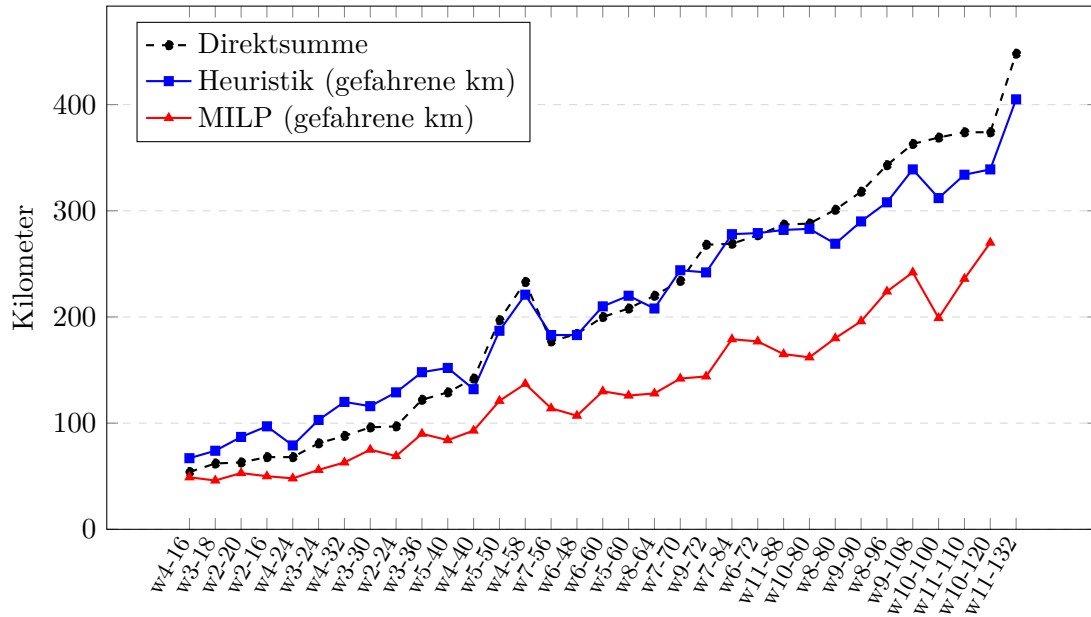
Für die erweiterten Instanzen wurden mit demselben Aufbau wie in Abschnitt 6.3 auch MILP-Referenzwerte berechnet (Zeitlimit 1800 s). Innerhalb des Zeitlimits beweist der Solver auf keiner erweiterten Instanz Optimalität, und die MIP-Gaps zwischen Incumbent und Schranke wachsen mit der Instanzgröße von 2–4 % auf über 16 %. Die bewiesenen oberen Schranken erlauben aber denselben konservativen Vergleich wie in Abschnitt 6.4. Der Gap der Heuristik gegen die Schranke bleibt dabei stabil: Er liegt überwiegend zwischen 12,6 und 18,1 % (Mittel 15,2 %), ohne erkennbaren Anstieg zur Instanzgröße. Der Ausreißer von 22,9 % auf w11-132 misst gegen eine besonders schwache Schranke (siehe unten).

Deutlicher ist der Befund bei der Akzeptanz: Ab etwa 84 Anfragen findet das MILP im Zeitlimit zunehmend keine Lösung mehr, die alle Anfragen bedient. Auf w7-84 bleiben drei Anfragen unbedient, auf w8-96 und w10-100 je eine, auf w11-110 vier und auf w10-120 neun. Die Heuristik bedient auf sämtlichen erweiterten Instanzen alle Anfragen. Auf w10-120 übertrifft sie den Incumbent damit erstmals auch im Zielwert (1235 gegenüber 1182): An dieser Größenordnung liefert das exakte Verfahren innerhalb des Zeitlimits die schlechtere Lösung. Auf der größten Instanz w11-132 gelang auf der verwendeten Hardware im Zeitlimit keine nutzbare MILP-Lösung: Allein der Modellaufbau benötigt rund 570 Sekunden, und im verbleibenden Zeitbudget fand der Solver nur die leere Lösung, die keine einzige Anfrage bedient. Die ausgewiesene Schranke ist entsprechend schwach. Auf leistungsfähigerer Hardware oder mit mehr Rechenzeit wäre hier voraussichtlich eine nutzbare Lösung erreichbar. Beim Kilometervergleich ist auf diesen Instanzen zu beachten, dass der Incumbent weniger Anfragen bedient und schon deshalb weniger fährt.

6.6 Einfluss der Gewichtsparameter

Die bisherigen Läufe verwenden die *customer-focused*-Gewichtung $w_1 = 10$, $w_2 = 1$. Dieser Abschnitt prüft, ob sich mit anderen Gewichten kürzere Strecken erreichen lassen, ohne die Akzeptanz zu opfern. Dazu wurde die Heuristik mit w_1 von 20 bis hinunter zu 1 (bei $w_2 = 1$) auf den Basis-Instanzen ausgeführt; Abbildung 6.3 zeigt die gefahrenen Kilometer aller Gewichtungen, Tabelle 6.12 die Gesamtbilanz.

Das Ergebnis ist eindeutig: Die Gewichtsänderung hilft nicht. Die Kurven in Abbildung 6.3 liegen über weite Strecken exakt aufeinander. Auf elf der 14 Instanzen ändern sich die Routen zwischen $w_1 = 6$ und $w_1 = 20$ überhaupt nicht. Das folgt aus dem Aufbau der Heuristik: Die beste Einfügeposition je Fahrzeug und die Regret-Reihenfolge hängen nur vom Zusatzfahraufwand ab, nicht von den Gewichten. Die Gewichte wirken allein über die Annahme- bzw. Aufschiebeschwelle $w_1 \cdot q_r / w_2$ aus Abschnitt 5.1.7. Kürzere Strecken entstehen deshalb ausschließlich durch *Ablehnungen*, und Tabelle 6.12 zeigt diesen Zusammenhang als streng monotonen Trade-off: Mit sinkendem w_1 fallen Akzeptanz und gefahrene Kilometer gemeinsam. Die Ersparnis wird erst ab etwa $w_1 = 3$ positiv, und bei $w_1 = 1$ bedient die Heuristik nur noch 43 % der Anfragen, dann allerdings mit nichtnegativer Ersparnis auf jeder einzelnen Instanz. Einen Gewichtspunkt, der volle Akzeptanz *und* kürzere Strecken liefert, gibt es nicht. Der einzige Fall mit vol-



Instanz (links Basis-, rechts erweiterte Instanzen, je aufsteigend nach Direktsumme)

Abb. 6.2: Gefahrene Kilometer von Heuristik (1aenge-Seeding, beste Variante) und MILP im Vergleich zur Direktsumme. Linker Block: die 14 Basis-Instanzen; rechts anschließend die 19 erweiterten Instanzen, jeweils aufsteigend nach Direktsumme sortiert. Die MILP-Kilometer stammen ab w3-36 aus dem Incumbent (Tabelle 6.10; erweiterte Instanzen: Abschnitt 6.5); auf den größten Instanzen bedient der Incumbent nicht alle Anfragen und fährt schon deshalb weniger.

ler Akzeptanz und Ersparnis ist w5-50 bei $w_1 = 8$ (ein Kilometer). Umgekehrt nimmt $w_1 = 20$ auch die letzte, bei $w_1 = 10$ unbediente Anfrage auf w2-20 an, für 13 Mehrkilometer dort. Die Straffung der Routen bei voller Akzeptanz bleibt damit strukturell verbesserten Verfahren vorbehalten.

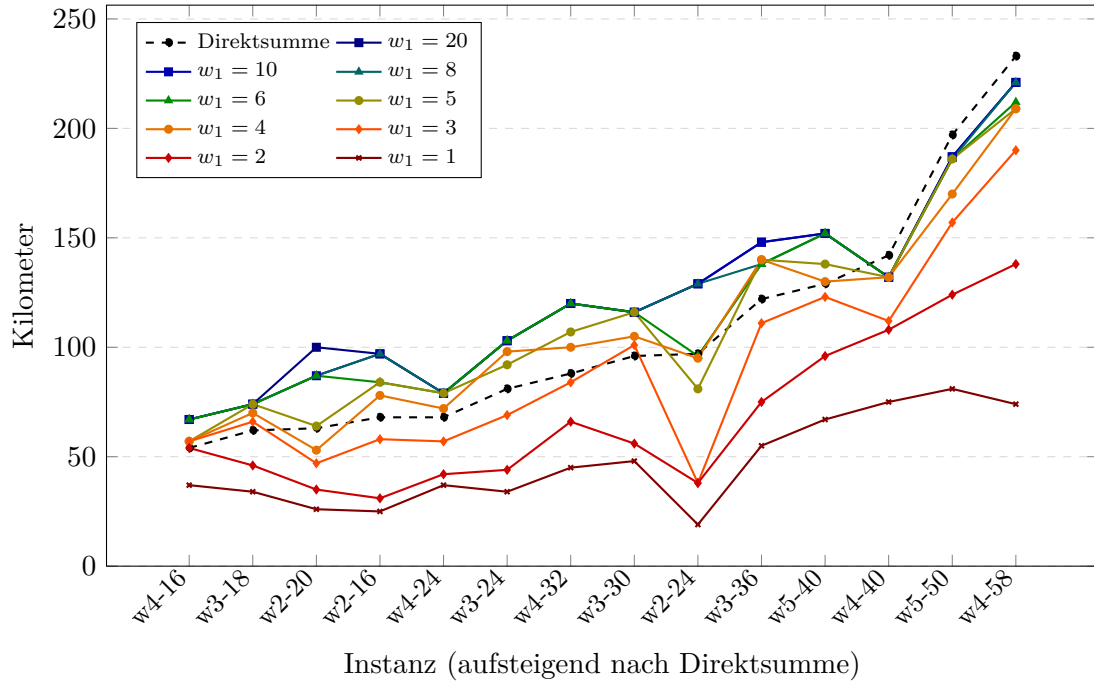


Abb. 6.3: Gefahrene Kilometer der Heuristik (*laenge*-Seeding, beste Variante) für alle untersuchten Gewichtungen w_1 von 20 bis 1 (bei $w_2 = 1$) auf den Basis-Instanzen. Für $w_1 \geq 8$ liegen die Kurven fast vollständig aufeinander; darunter sinken die Kilometer nur, weil zunehmend Anfragen abgelehnt werden (Tabelle 6.12).

Tab. 6.12: Gesamtbilanz über alle 14 Basis-Instanzen für verschiedene Gewichtungen w_1 (bei $w_2 = 1$). *Direkt (bedient)* = Summe der Direktstrecken der *bedienten* Anfragen; *Ersparnis* = Direkt (bedient) – km (gefahren), also die eingesparte Distanz aus Abschnitt 2.4 bezogen auf die tatsächlich bediente Nachfrage.

w_1	Akzeptanz [%]	$\sum$ Direkt (bedient)	$\sum$ km (gefahren)	$\sum$ Ersparnis
20	100,0	1500	1725	–225
10	99,8	1498	1712	–214
8	99,5	1496	1701	–205
6	97,9	1461	1646	–185
5	95,3	1423	1559	–136
4	93,2	1403	1509	–106
3	82,5	1254	1270	–16
2	65,2	1019	953	+66
1	43,5	795	657	+138

7 Diskussion

7.1 Myopie: gelöst bei der Auswahl, offen bei der Platzierung

Ein bekanntes Problem der Greedy-Insertion ist ihre Myopie (Abschnitt 4.2): Sie priorisiert die Anfrage mit dem aktuell geringsten Einfügeaufwand, ohne zu berücksichtigen, wie diese Entscheidung die Einplanungsmöglichkeiten der noch ungeplanten Anfragen einschränkt. Mögliche Folgen sind unnötige Ablehnungen und eine suboptimale Akzeptanzrate. Die Ergebnisse aus Kapitel 6 erlauben eine differenzierte Antwort darauf, ob und wo Regret- k dieses Problem tatsächlich behebt.

Dass die Myopie kein rein theoretisches Problem ist, zeigt die Greedy-Baseline unmittelbar: Ohne Seeding lehnt sie auf w2-16 drei, auf w2-20 sechs und auf w3-24 zwei Anfragen ab, auf der erweiterten Instanz w6-72 eine (Tabellen 6.1 und 6.11); frühe billige Einfügungen nehmen späteren Anfragen die letzten zulässigen Positionen. Regret- k mit **laenge**-Seeding bringt dagegen auf 32 der 33 Instanzen sämtliche Anfragen unter; einzig auf w2-20 bleibt eine Anfrage unbedient (Tabellen 6.9 und 6.10 sowie Abschnitt 6.5). Damit erreicht die Heuristik praktisch dieselbe Akzeptanz wie die exakte Referenz. Auf den größten Instanzen übertrifft sie sie sogar, weil der Incumbent dort im Zeitlimit selbst Anfragen unbedient lässt (Abschnitt 6.5). Die *Akzeptanz-Myopie* (unnötige Ablehnungen durch eine kurzsichtige Wahl der Einfügereihenfolge) ist auf dem untersuchten Benchmark damit nahezu vollständig behoben. Das ist der Effekt, für den Regret- k entworfen wurde: Anfragen, die nur noch in wenigen Fahrzeugen zulässig einfügbar sind, erhalten maximalen Regret und werden eingeplant, bevor ihre letzten Optionen verbaut sind.

Der verbleibende Abstand zum Optimum (Abschnitt 6.4) hat eine andere Ursache. Da beide Verfahren praktisch dieselbe Akzeptanz erreichen, entsteht er fast ausschließlich aus der Routeneffizienz, und die Kilometerspalten in den Tabellen 6.9 und 6.10 zeigen das deutlich: Das MILP unterschreitet die Summe der Direktstrecken auf jeder der 14 Basis-Instanzen, die Heuristik erst ab etwa 40 Anfragen (Abbildung 6.2). Dahinter steckt eine zweite Form der Kurzsichtigkeit, die Regret- k nicht adressiert und die sich als *Revisions-Myopie* bezeichnen lässt: Einmal eingefügte Anfragen werden nie wieder verschoben oder entfernt. Regret- k wählt klug aus, *welche* Anfrage als nächste eingeplant wird, aber jede Platzierung ist endgültig; frühe Einfügungen, die sich später als ungünstig herausstellen, bleiben in der Route und kosten Kilometer. Das MILP unterliegt dieser Einschränkung nicht, da es alle Einplanungen simultan optimiert. Naheliegende Abhilfen wie Removal-Reinsertion-Schritte im Stil der Large Neighborhood Search [RP06] würden hier ansetzen, allerdings zulasten der geringen Laufzeit.

Zusammengefasst: Regret- k beseitigt die Myopie der *Auswahl* (wen zuerst), nicht die Myopie der *Platzierung* (wohin endgültig). Für die Zielsetzung des liDARP mit ihrer starken Gewichtung der Akzeptanz ($w_1 = 10$) behebt es damit den wichtigeren Teil des

Problems.

7.2 Einordnung der Ergebnisse

Laufzeit. Auf den Basis-Instanzen läuft ein einzelner Regret-2-Durchlauf in 10 bis rund 670 Millisekunden (Tabelle 6.7). Auf den erweiterten Instanzen bleibt der Aufwand handhabbar: Ein Regret-2-Lauf benötigt auf der größten Instanz w11-132 rund sechs Sekunden, alle Varianten von Greedy bis Regret-11 zusammen etwa eine Minute. Das exakte Verfahren beweist Optimalität dagegen nur bis 32 Anfragen (dort bereits mit 908 Sekunden für w4-32) und läuft ab 36 Anfragen in das Zeitlimit von 1800 Sekunden. Der Laufzeitvorteil der Heuristik wächst also in dem Bereich, in dem er gebraucht wird.

Wann welches Verfahren? Auf kleinen Instanzen gibt es inhaltlich keinen Grund für die Heuristik: Das MILP fährt bei mindestens gleicher Akzeptanz deutlich weniger Kilometer; wer die Rechenzeit hat, nimmt das exakte Verfahren. Interessant wird die Heuristik ab der Größenordnung, in der der Optimalitätsbeweis scheitert (ab etwa 36 Anfragen): Dort liefert sie in Sekunden Lösungen mit voller Akzeptanz und einem konservativen Abstand von höchstens rund 18 % zur bewiesenen Schranke. Ab etwa 88 Anfragen unterschreitet sie zudem durchgehend die Direktsumme (Tabelle 6.11), erwirtschaftet also selbst eine Kilometerersparnis. Ab etwa 84 Anfragen verschiebt sich das Bild: Das MILP büßt im Zeitlimit zunehmend Akzeptanz ein, auf w10-120 liefert die Heuristik erstmals den besseren Zielwert als der Incumbent, und auf w11-132 liefert das MILP auf der verwendeten Hardware im Zeitlimit gar keine nichttriviale Lösung mehr (Abschnitt 6.5), bei Rechenzeiten der Heuristik von Sekunden gegenüber dem Zeitlimit von 1800 Sekunden.

Wahl von k . Eine klare Empfehlung für ein bestimmtes k geben die Ergebnisse nicht her: Der Bestwert stammt je nach Instanz von Regret-2 bis hin zu Regret-10, ohne erkennbares Muster (Tabellen 6.3 und 6.11); allenfalls liefern auf großen Flotten häufiger hohe Regret-Ordnungen das beste Ergebnis, und vereinzelt liegt sogar die Greedy-Baseline vorn (etwa w3-36, w5-50, w8-80). Praktisch ist das unproblematisch: Da alle Varianten zusammen selbst auf der größten Instanz nur rund eine Minute kosten, ist die sinnvolle Anwendung, sämtliche k von 2 bis κ zu rechnen und die beste Lösung zu übernehmen; genau so ist die Best-Spalte der Ergebnistabellen definiert.

Rolle des Seedings. Der Strategievergleich (Tabelle 6.6) zeigt vor allem, wie klein die Unterschiede sind: Zwischen der besten und der schlechtesten Strategie liegen gut 3 % der Summe der Bestwerte, und selbst ein zufälliges Seeding ordnet sich mitten in diesem Band ein und schlägt zwei der vier Strategien. Ein aufwendig gewähltes Seeding ist also keine Erfolgsvoraussetzung. Einzig **1aenge** zeigt Potenzial: Es liefert die höchste Summe der Bestwerte, gewinnt instanzweise am häufigsten und liefert auch beim Vergleich mit den exakten Lösungen die kleinsten Gaps (Abschnitte 6.2 und 6.4). Deshalb wird **1aenge** als Standard empfohlen. Zugleich ist das Seeding selbst ein Eingriff mit Nebenwirkungen: Da jedes Fahrzeug einen Seed erhält, werden die κ ausgewählten Anfragen zwangsweise

auf verschiedene Fahrzeuge verteilt. Auch dann, wenn zwei von ihnen in derselben Route besser aufgehoben wären oder eine kleinere Fahrzeugzahl genügen würde. Wie jede Einfügung ist auch ein Seed unwiderruflich; ungünstige Startverteilungen bleiben also bestehen und reihen sich in die in Abschnitt 7.1 beschriebene Revisions-Myopie ein.

Rolle der Gewichte. Über eine Änderung der Gewichtsparameter konnten in dieser Arbeit keine kürzeren Routen bei voller Akzeptanz erreicht werden (Abschnitt 6.6): Die untersuchten Gewichtungen verschieben lediglich den Punkt auf dem Trade-off zwischen Akzeptanz und Streckeneffizienz; kürzere Strecken traten nur gemeinsam mit Ablehnungen auf.

7.3 Limitationen

Die Ergebnisse dieser Arbeit gelten unter einigen Einschränkungen, die bei ihrer Interpretation zu beachten sind.

Erstens ist die Untersuchung rein *statisch*: Alle Anfragen sind vorab bekannt, und die Einplanung erfolgt einmalig. Ob sich die beobachteten Eigenschaften auf ein dynamisches Szenario mit eintreffenden Anfragen übertragen, in dem Insertion-Heuristiken wegen ihrer inkrementellen Arbeitsweise besonders attraktiv wären, bleibt offen.

Zweitens stützt sich die gesamte Evaluation auf ein einziges reales Liniennetz (die Würzburger Linie 6 als Distanzgrundlage) und auf Benchmark-Instanzen einer Quelle. Die beobachteten Schwellen (etwa die Kilometerersparnis ab rund 88 Anfragen) sind daher als Größenordnungen für dieses Setting zu lesen, nicht als allgemeingültige Konstanten.

Drittens sind die Vergleichswerte des exakten Verfahrens mit einer Julia/Gurobi-Implementierung der ereignisbasierten Formulierung entstanden, nicht mit der C++/CPLEX-Implementierung aus [RSS24] (Abschnitt 6.3). Die hier berichteten MILP-Laufzeiten sind deshalb nicht direkt mit den dort publizierten vergleichbar. Die Zielwerte selbst sind davon unberührt. Auf den fünf großen Basis-Instanzen und den erweiterten Instanzen liegen zudem nur bewiesene Schranken statt Optima vor, die ausgewiesenen Gaps sind dort konservative Obergrenzen.

Viertens ist die Heuristik in Python implementiert, das exakte Verfahren läuft auf einem hochoptimierten kommerziellen Solver. Der Laufzeitvergleich unterschätzt das Potenzial der Heuristik damit eher: In einer kompilierten Sprache wären die gemessenen Zeiten nochmals deutlich kleiner.

8 Fazit und Ausblick

8.1 Fazit

Ausgangspunkt dieser Arbeit war die Frage, ob sich das statische liDARP mit einer Regret- k -Insertion brauchbar lösen lässt, also ob das Verfahren Lösungen liefert, die in Akzeptanz, Routenqualität und Laufzeit einen praktischen Einsatz rechtfertigen. Dazu wurden die Regret- k -Insertion samt vier Seeding-Strategien sowie eine Greedy-Baseline implementiert und auf 33 Benchmark-Instanzen (16 bis 132 Anfragen) evaluiert, mit der ereignisbasierten MILP-Formulierung als Referenz.

Die zentralen Befunde: Mit **1aenge**-Seeding bedient die Heuristik auf 32 der 33 Instanzen sämtliche Anfragen (einzig auf w2-20 bleibt eine von 20 unbedient), während die Greedy-Baseline auf mehreren Instanzen ablehnt. Der Abstand zum Optimum beträgt auf den neun exakt gelösten Basis-Instanzen im Mittel 17,2 %; gegen die bewiesenen Schranken gemessen liegt er konservativ bei 14,4 % (große Basis-Instanzen) bzw. 15,2 % (erweiterte Instanzen). Er entsteht fast ausschließlich aus der Routeneffizienz, nicht aus Ablehnungen. Ab etwa 88 Anfragen unterschreiten die Routen der Heuristik durchgehend die Summe der Direktstrecken, das Verfahren erwirtschaftet dort also selbst eine Kilometerersparnis. Die Laufzeit bleibt dabei von Millisekunden auf den kleinen bis wenige Sekunden je Lauf auf den größten Instanzen, während das exakte Verfahren auf der verwendeten Hardware Optimalität nur bis 32 Anfragen beweist, ab etwa 84 Anfragen im Zeitlimit Akzeptanz einbüßt, auf w10-120 hinter den Zielwert der Heuristik zurückfällt und auf der größten Instanz (132 Anfragen) keine nutzbare Lösung mehr liefert. Eine eindeutig beste Regret-Ordnung k gibt es nicht; da alle Varianten zusammen nur geringe Rechenzeit kosten, ist die praktikable Anwendung, sämtliche k zu rechnen und die beste Lösung zu übernehmen.

Die Antwort auf die Ausgangsfrage fällt damit positiv aus, muss aber eingeschränkt werden. Die beiden Ziele des liDARP erfüllt Regret- k sehr unterschiedlich gut. Das Erschließungsziel (Mobilität für Menschen, die kein Auto besitzen oder nicht selbst fahren können) bedient es voll: Praktisch jede Anfrage wird angenommen, in Sekunden und auch auf Instanzgrößen, für die das exakte Verfahren auf der verwendeten Hardware im Zeitlimit keine nutzbare Lösung mehr liefert. Beim Umweltziel bleibt es dagegen deutlich hinter den exakten Lösungen zurück: Die Routen der Heuristik unterschreiten die Summe der Direktstrecken erst auf den größten Instanzen und auch dort nur knapp, während die MILP-Lösungen durch Bündelung durchweg erheblich Kilometer einsparen. Den Umweltvorteil des Ridepoolings kann Regret- k in dieser Form also kaum erzielen. Auch mit anderen Gewichtungen wurden in dieser Arbeit keine kürzeren Routen bei voller Akzeptanz erreicht (Abschnitt 6.6). Als Angebot für Menschen ohne Fahralternative ist das Verfahren trotzdem brauchbar, sobald die Instanzen für das exakte Verfahren zu

groß werden. Als ergänzender Befund zeigt die Analyse, woher dieser Rückstand kommt: Die Myopie der *Auswahl* (unnötige Ablehnungen durch eine kurzsichtige Einfügereihenfolge) behebt Regret- k auf dem untersuchten Benchmark bis auf eine einzelne Anfrage vollständig. Die Myopie der *Platzierung* (einmal eingefügte Anfragen werden nie revidiert) bleibt bestehen und erklärt den Rückstand in der Routeneffizienz. Sie markiert zugleich den natürlichen Ansatzpunkt für Verbesserungen.

8.2 Ausblick

Regret- k wurde in dieser Arbeit als eigenständige Konstruktionsheuristik untersucht. In der Literatur zu Pickup-and-Delivery-Problemen ist es darüber hinaus ein etablierter Baustein größerer Verfahren: In der Adaptive Large Neighborhood Search (ALNS) nach Ropke und Pisinger [RP06] dient die Regret-Insertion als *Repair*-Operator, der nach dem gezielten Entfernen von Anfragen (*Destroy*) die Lösung wieder vervollständigt. Die vorliegende Implementierung kann unverändert diese Rolle übernehmen und damit den Grundstein für eine ALNS für das liDARP legen; zu ergänzen wären Destroy-Operatoren und ein Akzeptanzkriterium. Ein solches Verfahren würde die in Abschnitt 7.1 beschriebene Revisions-Myopie direkt angehen, denn einmal platzierte Anfragen könnten wieder entfernt und neu eingefügt werden. Zu erwarten wäre ein kleinerer Kilometerrückstand gegenüber dem MILP; zum Preis der Laufzeit, die dann nicht mehr im Sekundenbereich läge.

Literaturverzeichnis

- [BRS25] Jonas Barth, Kendra Reiter und Marie Schmidt: The Line-Based Dial-a-Ride Problem with Transfers. In: *Proc. 25th Symp. Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'25)*, OpenAccess Series in Informatics (OASICS), 2025, 10.4230/OASICS.ATMOS.2025.17.
- [CL07] Jean François Cordeau und Gilbert Laporte: The Dial-a-Ride Problem: Models and Algorithms. *Annals of Operations Research*, 153(1):29–46, 2007, 10.1007/s10479-007-0170-8.
- [DD04] Marco Diana und Maged M. Dessouky: A New Regret Insertion Heuristic for Solving Large-Scale Dial-a-Ride Problems with Time Windows. *Transportation Research Part B*, 38(6):539–557, 2004, 10.1016/j.trb.2003.07.001.
- [GKS22] Daniela Gaul, Kathrin Klamroth und Michael Stiglmayr: Event-Based MILP Models for Ridepooling Applications. *European Journal of Operational Research*, 301(3):1048–1063, 2022, 10.1016/j.ejor.2021.11.053.
- [Hei24] Lucas Hein: Tabu Search für das linienbasierte dynamische DARP. Bachelorarbeit, Julius-Maximilians-Universität Würzburg, 2024.
- [HSK⁺18] Sin C. Ho, W.Y. Szeto, Yong Hong Kuo, Janny M.Y. Leung, Matthew Petering und Terence W.H. Tou: A Survey of Dial-a-Ride Problems: Literature Review and Recent Developments. *Transportation Research Part B*, 111:395–421, 2018, 10.1016/j.trb.2018.02.001.
- [LLMS21] Christian Liebchen, Martin Lehnert, Christian Mehler und Martin Schiefelbusch: Betriebliche Effizienzgrößen für Ridepooling-Systeme. In: *Wissenschaftsforum Mobilität 2020*, Universität Duisburg-Essen, 2021. Springer.
- [LMR⁺26] Antonio Lauerbach, Sven Mallach, Kendra Reiter, Marie Schmidt und Michael Stiglmayr: Solving the Line-Based Dial-a-Ride Problem by Generating Stopping Patterns, 2026. arXiv:2603.06295.
- [LRS25] Antonio Lauerbach, Kendra Reiter und Marie Schmidt: The Complexity of Counting Turns in the Line-Based Dial-a-Ride Problem. In: *SOFSEM 2025: Theory and Practice of Computer Science*, Lecture Notes in Computer Science, Seiten 85–98. Springer, 2025. doi: 10.1007/978-3-031-82697-9_7, arXiv:2409.15192.

- [MBC17] Yves Molenbruch, Kris Braekers und An Caris: Typology and Literature Review for Dial-a-Ride Problems. *Annals of Operations Research*, 259(1):295–325, 2017, 10.1007/s10479-017-2525-0.
- [PDH08] Sophie N. Parragh, Karl F. Doerner und Richard F. Hartl: A Survey on Pickup and Delivery Problems. Part II: Transportation Between Pickup and Delivery Locations. *Journal für Betriebswirtschaft*, 58(2):81–117, 2008, 10.1007/s11301-008-0036-4.
- [PR07] David Pisinger und Stefan Ropke: A General Heuristic for Vehicle Routing Problems. *Computers & Operations Research*, 34(8):2335–2351, 2007, 10.1016/j.cor.2005.09.012.
- [RP06] Stefan Ropke und David Pisinger: An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transportation Science*, 40(4):455–472, 2006, 10.1287/trsc.1050.0135.
- [RSS24] Kendra Reiter, Marie Schmidt und Michael Stiglmayr: The Line-Based Dial-a-Ride Problem. In: *Proc. 24th Symp. Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'24)*, Band 123 der Reihe *OpenAccess Series in Informatics (OASICS)*, 2024, 10.4230/OASICS.ATMOS.2024.14.
- [Ste77] David M. Stein: Scheduling Dial-a-Ride Transportation Systems: An Asymptotic Approach. Technischer Bericht TR-670, Division of Applied Sciences, Harvard University, Cambridge, MA, September 1977.

Titel der Bachelorarbeit:

Regret-k-Insertion für das statische linienbasierte Dial-a-Ride-Problem

Thema bereitgestellt von (Titel, Vorname, Nachname, Lehrstuhl):

Prof. Dr. Marie Schmidt

Eingereicht durch (Vorname, Nachname, Matrikel):

Albert Hörner, 2658264

Ich versichere, dass ich die vorstehende schriftliche Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die benutzte Literatur sowie sonstige Hilfsquellen sind vollständig angegeben. Wörtlich oder dem Sinne nach dem Schrifttum oder dem Internet entnommene Stellen sind unter Angabe der Quelle kenntlich gemacht.

Weitere Personen waren an der geistigen Leistung der vorliegenden Arbeit nicht beteiligt. Insbesondere habe ich nicht die Hilfe eines Ghostwriters oder einer Ghostwriting-Agentur in Anspruch genommen. Dritte haben von mir weder unmittelbar noch mittelbar Geld oder geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Arbeit stehen.

- ☒ Mit dem Prüfungsleiter bzw. der Prüfungsleiterin wurde abgestimmt, dass für die Erstellung der vorgelegten schriftlichen Arbeit Chatbots (insbesondere ChatGPT) bzw. allgemein solche Programme, die anstelle meiner Person die Aufgabenstellung der Prüfung bzw. Teile derselben bearbeiten könnten, entsprechend den Vorgaben der Prüfungsleiterin bzw. des Prüfungsleiters eingesetzt wurden. Die mittels Chatbots erstellten Passagen sind als solche gekennzeichnet.

Der Durchführung einer elektronischen Plagiatsprüfung stimme ich hiermit zu. Die eingereichte elektronische Fassung der Arbeit ist vollständig. Mir ist bewusst, dass nachträgliche Ergänzungen ausgeschlossen sind.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht. Ich bin mir bewusst, dass eine unwahre Erklärung zur Versicherung der selbständigen Leistungserbringung rechtliche Folgen haben kann.

Würzburg, 25.07.2026
Ort, Datum, Unterschrift

Erklärung zum Einsatz Künstlicher Intelligenz

Beim Erstellen dieser Arbeit Künstliche Intelligenz als Hilfsmittel eingesetzt. Diese Arbeit wurde einer Rechtschreibprüfung unterzogen und einzelne Formulierungen wurden korrigiert. Kein einzelner Absatz der Arbeit ist unverändert KI-generiert. Der Einsatz wird stattdessen im Folgenden je Arbeitsphase und Kapitel offengelegt.

Vorbereitung. Für die Einarbeitung in das Thema wurde neben dem eigenen Lesen der zugrunde liegenden Paper KI eingesetzt, um diese besser zu verstehen: Zusammenhänge wurden erklärt und Definitionen von Fachbegriffen erfragt.

Kapitel 1 (Einleitung). Vollständig ohne Einsatz von KI verfasst.

Kapitel 2 (Grundlagen). KI wurde zur Fehlersuche im Text und zum Korrekturlesen der Formeln verwendet.

Kapitel 3 (Verwandte Arbeiten). KI wurde für die ergänzende Literaturrecherche verwendet, neben Portalen wie Google Scholar und den vom Lehrstuhl bereitgestellten Arbeiten. Um Halluzinationen zu vermeiden, wurden ausschließlich Quellen aufgenommen, die im Volltext vorlagen und geprüft werden konnten.

Kapitel 4 (Methodik). KI wurde verwendet, um zu prüfen, ob unaufgedeckte Randfälle vorhanden sind, sowie zur Verifikation der mathematischen Formeln.

Kapitel 5 (Implementierung). KI wurde zum Abgleich zwischen Quellcode und Beschreibung verwendet.

Kapitel 6 (Evaluation). KI wurde zum inhaltlichen Abgleich mit den Ergebnissen der Experimente und dem Inhalt der Tabellen verwendet.

Kapitel 7+8 (Diskussion + Fazit und Ausblick). Keine Verwendung zur Bewertung der Ergebnisse